Responsive Web Design

DECAP784

Edited by
Dr. Amit Sharma

LOVELY PROFESSIONAL UNIVERSITY

Responsive Web Design

Edited By:
Dr. Amit Sharma

Content

Dr. Navneet Kaur, Lovely Professional University

Unit 01:Getting started with Responsive Web Design

CONTENTS

Objectives

After studying, you will be able to:

- Learn the basics of responsive web design
- Understand the pros and cons of responsive web design
- Understand the pros and cons of fluid width design
- Understand and implement fluid width design using CSS

Introduction

Responsive web design is a process of designing and building websites to provide better accessibility and optimal viewing experience to the user by optimizing it for different devices. With the growing trend of smart phones and tablets, it has become almost unavoidable to ignore the optimization of sites for mobile devices. Responsive web design is a preferable alternative and an efficient way to target a wide range of devices with much less efforts. Responsive layouts automatically adjust and adapts to any device screen size, whether it is a desktop, a laptop, a tablet, or a mobile phone. Figure 1 shows the Illustration.

Figure 1

1.1 Understanding Responsive Web Design

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities.With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim. How is one to design for these situations?Refer to Figure 2.

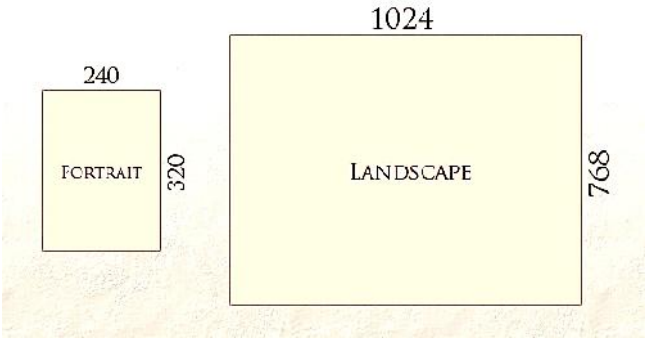


Figure 2

Why Responsive Web Design Is Important

There was a time when people only accessed websites from a desktop computer. The vast majority had the same size monitor. Websites were designed for the average visitor. Today, people access websites from a variety of different devices with screens ranging from a few inches all the way up to 27" or more and expectations have changed. Consumers expect the website they're visiting to know that they're using a tablet vs. a PC. They expect the site to adjust to them—not the other way around.Different devices also come with different expectations in terms of usability. For example, the Apple iPhone has "trained" people to expect to be able to swipe left/right/up/down. People visiting a website from a smartphone expect to be able to click a phone number and have their phone give them the option of auto-dialing that number. Similarly, they expect an address to have a "click for directions" option that uses their phone's GPS. A site that is responsive takes all of this into account and automatically adjust to provide visitors with the best possible user experience regardless of the device being used to access the site.

How does responsive web design work?

Responsive web design works through Cascading Style Sheets (CSS), using various settings to serve different style properties depending on the screen size, orientation, resolution, color capability, and other characteristics of the user's device. A few examples of CSS properties related to responsive web design include the viewport and media queries.

How to check if a website is responsive?

You can quickly see if a website is responsive or not in your web browser.

1. Open Google Chrome
2. Go to your website
3. Press Ctrl + Shift + I to open Chrome DevTools
4. Press Ctrl + Shift + M to toggle the device toolbar
5. View your page from a mobile, tablet, or desktop perspective

You can also use a free tool, like Google's Mobile-Friendly Test, to see if pages on your website are mobile-friendly. While you can achieve mobile-friendliness with other design approaches, such as adaptive design, responsive web design is the most common because of its advantages.

1.2 Pros of Responsive Web Design

There are many advantages(Figure 3) of RWD:

1. **User experience friendly:** Responsive web design is specially designed to respond according to customer's or user's behaviors and their needs, according to screen size, etc. It is used to create a website that adjusts smoothly to different screen sizes especially for mobile viewing. Therefore, it increases reach to customers and users on small devices like mobiles, tablets, etc.
2. **Save cost on responsive web design development:** Comparing with the development for websites on PC, iPad, and mobile phone, the responsive design is more conducive to saving design and development costs.In terms of design, it only needs to design two sets of design renderings for the responsive web interface based on the PC, iPad, and mobile. From the perspective of the front-end development, itonly needs to develop 3 different sets of CSS styles. From the perspective of post-maintenance, there is no need to maintain PC interface, iPad interface, and mobile interface.
3. **SEO friendly:** Responsive websites are generally responsible for ranking top at SEO (Search Engine Optimization). On mobile devices or small devices, a responsive website loads much faster as compared to a desktop or laptop. This automatically increases positive user experience that in turn gives a higher ranking to the website on SEO.
4. **Increase profit and sale:** Responsive web design is easy to create and faster to implement. This is because there is no requirement for another website for small devices. A responsive website is specially designed to fit all screen sizes. Therefore, one can save time, effort, maintenance costs and development cost associate with creating another website for small devices. It generally increases user experience and reaches more audiences through small devices. More will be an audience, more will be profit and sale.
5. **Low maintenance:** Responsive websites are designed to fit all screen sizes. There is no change in content and other elements to fit on a different device. This website uses the same content across all devices. There is only one website that fits all screen sizes so the cost of maintaining two websites is also saved because maintaining a separate site for a small device requires a lot of testing and support. Managing a single website requires less maintenance, less cost, saves time, etc.
6. **Easy to track users:** Responsive websites take less time to load, saves time, money, save development effort of creating another website for small devices. One can use this time and effort to track their site visitors.

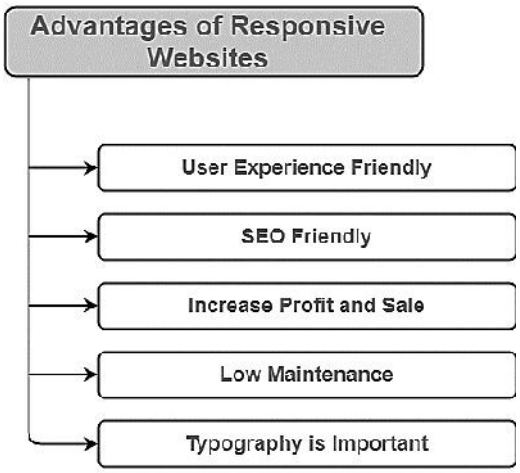


Figure 3

1.3 Cons of Responsive Web Design

There are many disadvantages(Figure 4) of RWD:

1. **Bad compatibility for the old version of IE browser:**If your site users are mostly using the old version of IE, it is not recommended to do responsive design. That's a fatal problem for the older version of IE browser (IE6, IE7, IE8).
2. **Slow page loading:** Though, responsive websites are easy to maintain, but it sometimes takes a long time to load the page. It includes some high-resolution images and videos that sometimes require a lot of time to load.
3. **Navigation is tough:** Responsive websites are specially designed to fit on small devices. But maintaining the simplicity of large websites for small devices sometimes becomes difficult. This is because small devices have less screen size and this turn make it more difficult to navigate website through small devices.
4. **Time-consuming development:** Responsive websites are essential but take a lot of time in the development process as compared to the development time of normal websites.

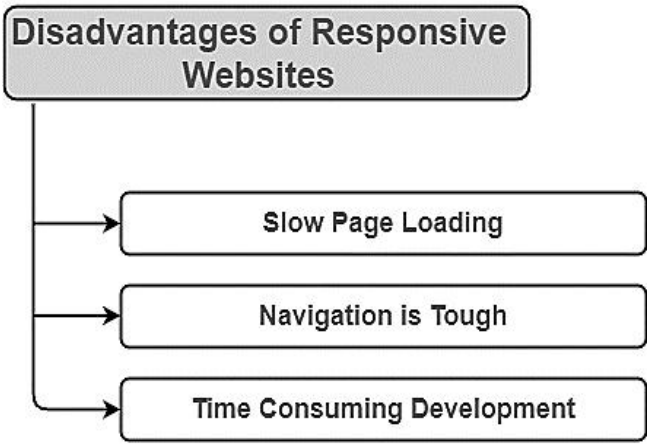


Figure 4

1.4 Fluid or Elastic or Liquid Layout

Most webpage layouts include one, two, or three columns. In the early days of web design, when most users had similar screen sizes, web developers would assign the columns fixed widths. For example, a fixed layout may include a main content area that is 960px wide with three columns that have widths of 180px, 600px, and 180px. While this layout might look great on a 1024x768 screen, it might look small on a 1920x1080 screen and would not fit on a 800x600 screen.Fluid layouts solve this problem by using percentages to define each area of the layout. For example, instead of creating a content area of 960px, a web developer can create a layout that fills 80% of the screen and the three columns could take up 18%, 64%, and 18% respectively. By using percentages, the content can expand or shrink to fit the window of the user's computer. The CSS used to create a fixed layout vs a fluid layout is shown below.

Fixed layout	Fluid layout
.content { width: 960px; }	.content { width: 80%; }
.left, .right { width: 180px; }	.left, .right { width: 18%; }
.middle { width: 600px; }	.middle { width: 64%; }

The CSS classes in the examples could each be assigned to a div within a page's HTML where the .left, .right, and .middle classes are enclosed within the .content class. The content class could also be a assigned to a table and the other classes could be assigned to table cells. The fixed width .content class does not require a defined width since it automatically spans the width of the enclosed divs or table cells.

So, a fluid layout is a type of webpage design in which layout of the page resizes as the window size is changed. This is accomplished by defining areas of the page using percentages instead of fixed pixel widths.In a fluid website layout, also referred to as a liquid layout, the majority of the components inside have percentage widths, and thus adjust to the user's screen resolution.It is a layout in which the width of main container is flexible(in percentage). Whatever the screen-size is, the layout will remain same. Refer to Figure 5. The image in Figure 5 shows a fluid (liquid) website

layout. While some designers may give set widths to certain elements in fluid layouts, such as margins, the layout in general uses percentage widths so that the view is adjusted for each user.

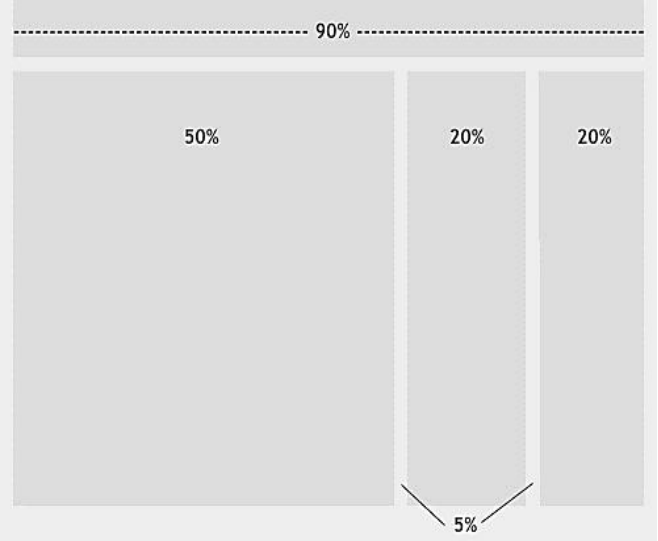


Figure 5

Properties of Fluid Layout

1. Width is in percentage.
2. Text scroll down when zoom in or zoom out
3. Dependent of screen size.
4. Horizontal Scroll will never come on any screen unless any fixed content is inside.

Designers may not use fluid page designs for various reasons, but the layout's benefits are often overlooked. Below are pros and cons to think about when considering fluid web page design.

Pros

1. Fluid web page design can be more user-friendly, because it adjusts to the user's set up.
2. The amount of extra white space is similar between all browsers and screen resolutions, which can be more visually appealing.
3. If designed well, a fluid layout can eliminate horizontal scroll bars in smaller screen resolutions.

Cons

1. The designer has less control over what the user sees and may overlook problems because the layout looks fine on their specific screen resolution.
2. Images, video and other types of content with set widths may need to be set at multiple widths to accommodate different screen resolutions.
3. With incredibly large screen resolutions, a lack of content may create excess white space that can diminish aesthetic appeal.

Below is the example:

```
<!DOCTYPE html>
<htmllang="en">
<head>
<meta charset="UTF-8">
```

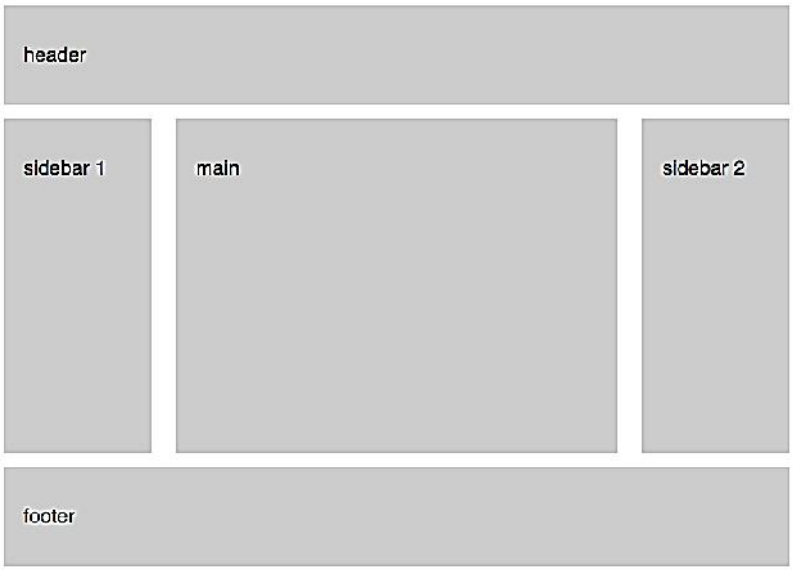
```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>

<style>
    *{
        margin:0;
box-sizing:border-box;
    }
    .container{
        width:80%;
margin:auto;
    }
    header{
        padding:10px;
background:aqua;
    }
    nav{
        padding:10px;
background:gray;
    }
    main{
display:flex;
    }
    aside{
background:pink;
        flex: 1 0 25%;
        padding:10px;
    }
    section{
background:yellow;
        flex: 1 0 75%;
        padding:10px;
    }
    footer{
        padding:10px;
        background:#333;
        color:#fff;
    }

</style>
```

```
</head>
<body>
<div class="container">
<header>This is header</header>

<nav>navigation menu</nav>

<main>
<aside>this is left sidebar</aside>
<section>this is right section</section>
</main>

<footer>this is footer</footer>
</div>
</body>
</html>
```

The output of above code is shown in Figure 6.

Figure 6

Task: Develop the following web page using fluid width design



Difference between fixed and Fluid layouts

Property	Fixed layout	Liquid layout
Width of wrap	In Pixels (960px, 1200px).	In Percentage or auto, for exp 80%
Height	in Pixels or auto.	Automatic
Device Compatibility	For devices greater than their width.	Remain Same. Do not compress
Text Scrolling on various Devices	Remain same	Text scroll down
Print Friendly	Yes	No

Fluid Layout vs Responsive Design

The terms "fluid layout" and "responsive web design" are sometimes used interchangeably, but they are two different things. A page created using responsive web design includes CSS media queries, which load different styles depending on the width of the window or the type of device used to access the page. Responsive web design requires more CSS (and sometimes JavaScript) than a basic fluid layout, but it also provides more control over layout of the page.

Summary

1. Responsive design allows your website content to flow freely across all screen resolutions and sizes, and renders it to look great on all devices.
2. A fluid design covers the entire browser window by specifying the width in percentages rather than pixels.

Keywords

- RWD
- Fluid width

Self Assessment

1. RWD is screen dependent. This statement is ___.

A. True
B. False

2. In RWD, the website should not transition to account for resolution, picture size, and scripting capabilities as the user moves from their laptop to an iPad.

A. True
B. False

3. Most commonly used layout in CSS is/are:

A. Fixed width
B. Fluid width
C. Both
D. None

4. (i) RWD is SEO friendly
(ii) RWD involves low maintenance costs

(iii) RWD reduces the profit.
Choose the correct option
A. (i), (ii), and (iii) are correct
B. Only (ii) is correct
C. (i) and (iii) are correct
D. (i) and (ii) are correct

5. RWD has a bad compatibility with which of the following browsers?
A. Google Chrome
B. Safari
C. Internet Explorer
D. Mozilla

6. Fluid layout is also called as ___.
A. Liquid layout
B. Water layout
C. Both
D. None

7. In a fixed layout, the width is mentioned in ____.
A. px
B. percentage
C. Both
D. None

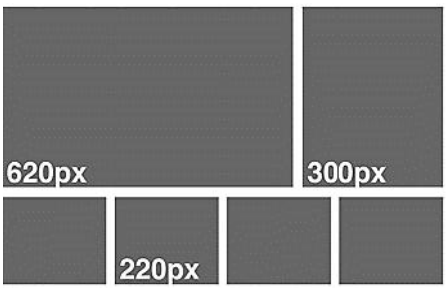
8. In a fluid layout, the width is mentioned in ____.
A. px
B. percentage
C. Both
D. None

9. Identify the type of layout in the given code:
.container{ width: 1000px; }
A. Fixed layout
B. Fluid layout

10. Identify the type of layout in the given code:
.container{ width: 60%; }
A. Fixed layout
B. Fluid layout

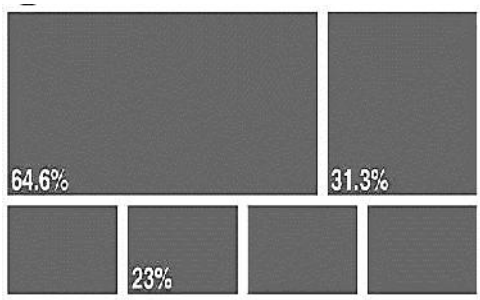
11. Fixed layout is more user friendly as compared to Fluid layout
A. True
B. False

12. Identify the kind of layout from the given picture



A. Fixed
B. Fluid

13. Identify the kind of layout from the given picture



A. Fixed
B. Fluid

14. ____ layout reduces user scrolling
A. Fixed
B. Fluid

15. ____ approach closely mimics print layout
A. Fixed layout
B. Fluid layout

Answers for Self Assessment

l.	A	2.	B	3.	C	4.	D	5.	C
6.	A	7.	A	8.	B	9.	A	10.	B
11.	B	12.	A	13.	B	14.	B	15.	A

Review Questions

1. What is a responsive web design? What is its purpose?
2. What are the different advantages of responsive web design? Explain briefly.
3. What are the different disadvantages of responsive web design? Explain briefly.
4. What is fluid width design? State its properties.
5. What are advantages of fluid width design?
6. What are disadvantages of fluid width design?
7. What is a fixed width design?
8. What is the difference between a fixed width design and a fluid width design?
9. Is RWD and fluid width one and the same concept? Explain.
10. Give an example explaining the concept of fixed width design

Further Readings

https://blog.hubspot.com/website/fluid-design

https://www.educative.io/answers/what-is-fluid-design

Unit 02:Getting Started with Responsive Web Design

CONTENTS

Objectives

After studying, you will be able to:

- Understand pixels and CSS units
- Understand and implement width= the target/context formula

Introduction

Modern content needs to be ready for a variety of viewing environments: smart phones, tablets, large monitors or even TV screens cover a huge range of sizes, aspect ratios, pixel densities and viewing distances. A number of tools are available to help developers optimize their layout for the best experience

 E.g., to avoid or minimize awkward scrolling.

2.1 Understanding Pixels and other CSS Units

In CSS, a variety of different units may be used to indicate length and measurement. The property size that we specify for an element, or its content is measured in a CSS unit. A growing number of CSS length units have provided new flexibility to web authors. For example, the 'rem' (root 'em') unit permits the font size of the root element to be used for sizing throughout the document. They help developers lay out content independently of display size and resolution.

Relative length units

A relative length takes sizing relative to some other thing, and therefore the final size of something defined using a relative length may be different if the thing it is relative to changes. The complete set of relative units is as follows. The first four units are font relative, while the last four are viewport relative.

Unit	Description	Example use case

em	1 em is the computed value of the font-size on the element on which it is used.	For example, the font size <h1> heading elements may be set to 3em and the body kept at 1em, making sure that under all display conditions heading text will be 3 times as large as the body's. It must be noted that when used as a font-size property value, the em unit refers to the font size of the parent element. Thus, in our example, a element inside an <h1> with font-size: 2 em would end up with text 6 times larger than in the body.
ex	1 ex is the current font's x-height. The x-height is usually (but not always, e.g., if there is no 'x' in the font) equal to the height of a lowercase 'x'	Rarely used in practice. May be used to size inline images to fit the x-height of the current font for visual harmony.
ch	1 ch is the advance of the '0' (zero) glyph in the current font. 'ch' stands for character.	Can be used to style monospace text or braille.
rem	1 rem is the computed value of the font-size property for the document's root element. This unit is often easier to use than the 'em' unit because it is not affected by inheritance as 'em' units are.	For example, given a root element font-size of 20px, setting a 0.5em font-size on elements would resolve to 10px for first-level but second-level would have a 5px font-size. Setting the font-size to 0.5rem would result in 10px elements no matter their nesting level.
vw	1vw is 1% of the width of the viewport. 'vw' stands for 'viewport width'.	Useful to size boxes that adapt to different viewport widths.
vh	1vh is 1% of the height of the viewport. 'vh' stands for 'viewport height'.	Useful to size boxes that adapt to different viewport heights. For example, may be used to set a maximum height on an image so that it does not exceed the viewport dimensions.
vmin	Equal to the smaller of 'vw' or 'vh'. 1vmin = 1% or 1/100 of the viewport's smaller dimension.	-
vmax	Equal to the larger of 'vw' or 'vh'. 1vmax = 1% or 1/100 of the viewport's larger dimension.	-

As these values are relative to something, it is important to identify exactly what they are relative to. For the font relative unit rem then this is always relative to the size of the root element which is an HTML document is the html element. In the first example below, we have set the html element to have a font-size of 20 pixels. 1rem is therefore 20 pixels. If we then give an element a width of 10rem, it will become 200 pixels wide (as 20px multiplied by 10 is 200).

When the other font relative units (em, ex, and ch) are used for the length of an element, they are relative to the font size as applied to that element. In the second example (the width of the box is 10em), the em unit looks at the font applied to the element which it is sizing and calculates based on that. So, this box becomes 300 pixels wide as the font-size of the box is 30px.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
   div { border-radius: 5px; margin-bottom: 20px;}

html {
  font-size: 20px;
}

.example-rem {
  width: 10rem;
font-size: 30px;
border: 2px solid rgb(88,104,164);
}

.example-em {
  width: 10em;
  font-size: 30px;
  border: 2px solid rgb(250,153,20);
}
</style>
</head>
<body>
<div class="example-rem">I am 10rem</div>

<div class="example-em">I am 10em</div>
</body>
```

```
</html>
```

The output of above code is shown in Figure 1.

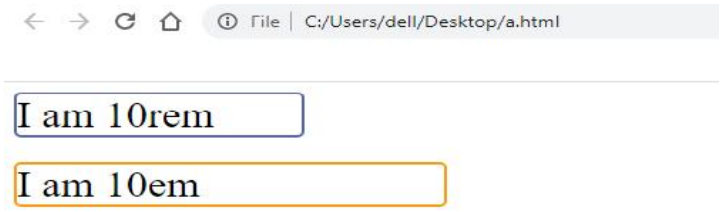


Figure 1

Where font relative units are calculated from font size, the viewport relative units are calculated in relation to a rectangle known as the initial containing block. On a screen, this has the dimensions of the viewport. The vw unit is 1/100 of the width of the viewport and vh 1/100 of the height. A box which has a width of 50vw and a height of 50vh should be half the width and half the height of the viewport.

The vmin and vmax units are useful because they allow you to size something relative to the larger or smaller dimension of the viewport. This means that you can make something 50% of the longest side of the viewport for example. This is especially helpful when someone might hold a device in landscape or portrait mode. The vmin unit always resolves to the small or vw or vh and vmax to the larger of vw or vh. Therefore, if you want a width to always be 20% of the longest side of the device, you can use 20vmax. If the device is held in portrait mode, then 20vmax would be the same as 20vh. If the device is held in landscape mode, it would be the same as 20vw.

The example below compares a block sized with vw and vh with one sized using vmin and vmax. On a desktop computer, or a phone in landscape mode, both boxes should appear the same size. Switch a phone to portrait mode or drag your window so that the width becomes smaller than the height, and you will see how the second block changes the dimension from which it takes the calculation. Refer to Figure 2 and Figure 3.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
   * { box-sizing: border-box; }
.box {
  border: 2px solid rgb(88,104,164);
  background-color: rgba(88,104,164,.2);
  border-radius: 5px;
```

```
}
.a {
  width: 50vw;
  height: 50vh;
}
.b {
  width: 50vmax;
  height: 50vmin;
}
</style>
</head>
<body>
<div class="box a">width: 50vw;<br>height: 50vh;</div>
<div class="box b">width: 50vmax;<br>height: 50vmin;</div>
</body>
</html>
```

Figure 2 Output on a desktop computer

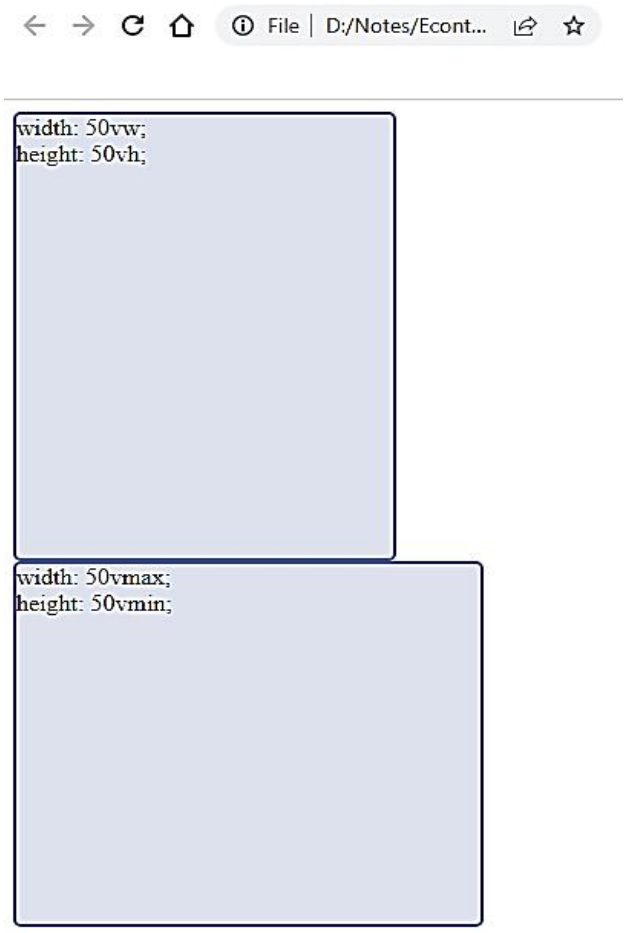


Figure 3 Output when the size of window is reduced

Absolute Units

The absolute units map to physical dimensions and do not scale relative to other things on the screen. Therefore, they are most useful when the output environment is known.The list below shows the allowed absolute units in CSS:

cm - It is used to define the measurement in centimeters.

mm - It is used to define the measurement in millimeters.

in - It is used to define the measurement in inches. 1in = 96px = 2.54cm

pc - It is used to define the measurement in picas. 1pc = 12pt so, there 6 picas is equivalent to 1 inch.

pt - It is used to define the measurement in points. 1pt = 1/72 of 1 inch.

px - It is used to define the measurement in pixels. 1px = 1/96th of inch

Percentage Units

Percentages allow the sizing of elements relative to their containing block. For example, we can set up the body of a document like so

```
body {
    width: 80%;
    max-width: 900px;
```

```
    margin-left: auto;
    margin-right: auto;
}
```

To ensure the body is at most 900px and take 80% of the width of the viewport otherwise. (Note that CSS pixels are not device pixels)

In most cases, you can use a percentage rather than a length unit for size. This percentage will then need to be calculated in relation to something, in the same way that a relative length unit is resolved, and the specification for the layout method you are using will indicate what the percentage should be a percentage of.

In a specification, where you see <length-percentage> as an allowable value for a length, this means that the percentage will be resolved to a length before being used. In the below example, the outer element has a width of 400 pixels, and the first child element a width of 50%. This then resolves to 200 pixels - 50% of 400.

The second child element has a width which uses calc, to add 50 pixels to 50%, making that block 250 pixels wide. The 50% is therefore resolved to a length and then used in the calculation.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
   html { font-size: 30px;}

div {
  border-radius: 5px;
}

.wrapper {
  width: 400px;
  border: 2px dotted rgb(245,199,28);
}

.a {
  width: 50%;
  border: 2px solid rgb(88,104,164);
}
```

```
.b {
  width: calc(50% + 50px);
  border: 2px solid rgb(250,153,20);
}


</style>
</head>
<body>
<div class="wrapper">
<div class="a">50%</div>
<div class="b">50% + 50px</div>
</div>
</body>
</html>
```

The output of above code is shown in Figure 4.

Figure 4

2.2 width= the target/Context Formula

The formula target ÷ context = result is used to convert the fixed (absolute) widths of elements on a page, normally expressed in pixels, to relative widths, widths expressed as a proportion of their container, normally expressed as a percentage. This allows us to create a webpage with a fluid grid design, opposed to the original fixed width design. The advantage of this, is that as a browsers width changes size (or a website is viewed on a screen with a different resolution) the elements on a page should scale appropriately to keep its design proportional to the original.

This is a required step of Responsive Web Design (RWD), which normally takes three steps:

1. Fluid Images - Use CSS Rule img:{max-width:100%;}. This ensures images will fill their parent element
2. Fluid Grid - Page elements set as relative widths using target ÷ context = result to ensure they scale appropriately.
3. Media Queries - To create "breakpoints" to enable you to change the layout of the page to be more suited for larger/smaller screens, rather than just allowing the fluid grid to continuing scaling the elements to either a ridiculously large or size or an unreadable small size.

The two most common instances you would use them are,

To convert pixels to em or rem.

Lets say you needed to convert a font size of 55px to em's.

Target= 55px

Context= 16px (the default base font size for most browsers)

55 ÷ 16 = 3.4375

Answer = 3.4375em

The second example would be getting a percentage of something.

Lets say you had a image that was 400px wide, your site has a width of 960px and you wanted to give your image a max width in %.

Target = 400px (image width) Context = 960px (website width)

400 ÷ 960 = 0.41666667

When working out the % we have to move the decimal point two places to the right to get our answer.

Answer = Image width is 41.666667%

Target: is the width of the element you want to convert.

Context: is the elements container width.

Result: will therefore be the size of the target expressed as a fraction of the context width.

So in the below image we have a single blue rectangle with a width of 300 pixels. It is surrounded by nothing other than the page itself, which by default will be 100% the width of the browser viewport, which in this scenario is 960 pixels. So let's use the formula on this blue rectangle element:

300px ÷ 960px = 0.3125

Now we need to convert this into a percentage to be able to use it as a relative width, this is simply done multiply by 100:

0.3125 * 100 = 31.25%

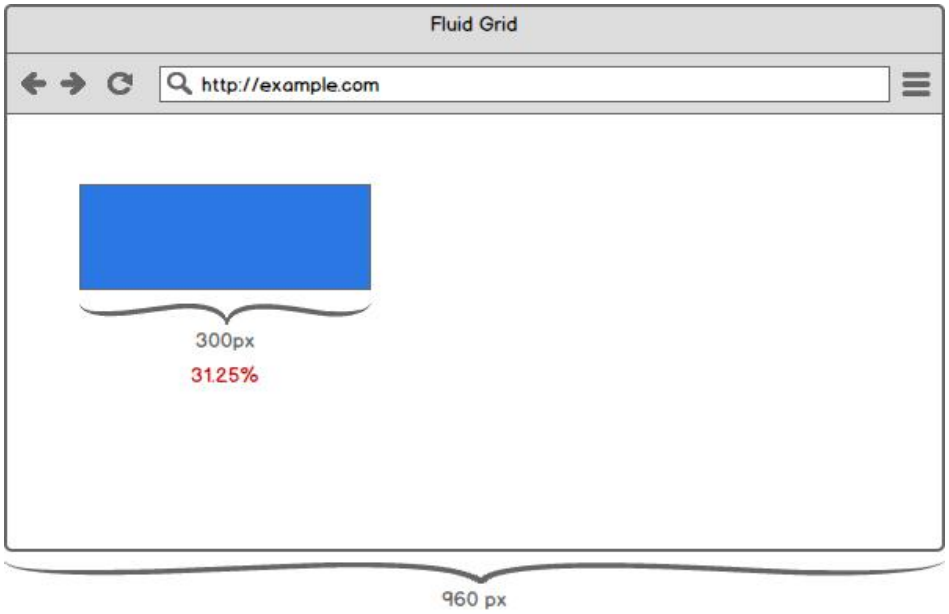


Figure 5

So now we change the width of our blue rectangle to 31.25%. Initially we shouldn't see a change, however when we resize the browser window, we should see that the blue rectangle width changes taking up 31.25% of the width of the browser viewport, therefore keeping its size proportional to the original design.

You can also apply this formula to the elements margins (or other positioning properties) so that its position remains proportional.

Summary

- A CSS unit determines the size of a property you're setting for an element or its content.
- The various units include relative length, absolute and percentage units

Keywords

CSS units

Self Assessment

1. em is a ___ unit.

A. Length unit
B. Absolute unit
C. Percentage unit
D. None

2. rem is root element

A. True
B. False

3. 1vw is ___% of the width of the viewport.
A. 100
B. 10
C. 1
D. None

4. vw stands for ___.
A. View width
B. Viewport width
C. Variable width
D. None

5. ____ is equal to the smaller of 'vw' or 'vh'
A. vmax
B. vh
C. vw
D. vmin

6. ____ is equal to the larger of 'vw' or 'vh'
A. vmax
B. vh
C. vw
D. vmin

7. Calculate the required percentage. Given, Target = 200px, Context = 960px
A. 20.83%
B. 4.65%

8. Calculate the target. Given, Result = 65%, Context = 960px
A. 14.76px
B. 624px

9. Which of the following is correct?
A. Result = context/target
B. Result = target x context
C. Result = target/context
D. None

10. vmin uses the ratio of the ___ side
A. smallest
B. largest

11. vmax uses the ratio of the ___ side
A. smallest
B. largest

12. Which of the following is an absolute unit?
A. rm
B. px
C. rem
D. ch

13. Which is of the following is a relative length unit?
A. pt

B. em
C. vh
D. px

14. A ___ takes sizing relative to some other thing.
A. Relative length
B. Absolute

15. The ____ units map to physical dimensions and do not scale relative to other things on the screen.
A. Relative
B. Absolute

Answers for Assessment

l.	A	2.	A	3.	C	4.	B	5.	D
6.	A	7.	A	8.	B	9.	C	10.	A
11.	D	12.	B	13.	D	14.	A	15.	B

Review Questions

1. Describe various length units in CSS
2. Explain the difference between vmin and vmax with the help of an example
3. Explain the difference between vh and vw with the help of an example
4. Explain the difference between rem and em with the help of an example
5. Give an example explaining the concept of percentage unit in CSS
6. Explain the formula width= the target/context with the help of an example.
7. What are the different absolute units in CSS.
8. What is the difference between relative length and absolute unit.
9. Describe with an example ex unit.
10. What is viewport width?

Further Readings

https://www.javatpoint.com/css-units

Unit 03:HTML5 structure for Website

CONTENTS

Objectives

After studying, you will be able to:

- Understand HTML page structure
- Understand and implement CSS resets

Introduction

An HTML 5 document mainly consists of a Head and Body. The Head contains the data, which informs the browser and even web servers that it is an HTML 5 document. On the other hand, the Body contains content that web browsers actually display. This chapter covers all the major elements necessary to craft a basic HTML 5 document.

3.1 HTML5 Page Structure

Each and every HTML 5 document employs a unique combination of elements and content to define a page. The structure of all the property documented pages is the same and contains:

- A declaration at the top, which indicates that it is an HTML 5 document
- A document header
- A document body

A collection of HTML 5 elements constitutes an HTML 5 document. Some of these elements are essential while others are optional. However, you can always find the following three elements on every page in addition to the DOC Type declaration at the top.

Version Information – Doctype

A basic HTML page starts with the Document Type Declaration or doctype. That is a way to inform the browser what type of document it is. The doctype is always the first item at the top of any HTML file. Then sections and subsections come, each possibly has its heading and subheading. These heading and sectioning elements help the reader to perceive the content meaning.

```
<!Doctype html>
```

The doctype can be written in lowercase, uppercase, or mixed case. As you noticed, the "5" is missing from the declaration. Although this web markup is known as "HTML5".

The <html> element

The <html> element follows the doctype information, which is used to inform the browser that this is an HTML document. You can use the lang attribute with the "en" value to specify that the document is in English. But nowadays, even the lang attribute is unnecessary for the document to validate or function correctly.

Don't forget to include the closing </html> tag:

```
<!DOCTYPE HTML>
<htmllang="en">


</html>
```

The <head> section

The next part is the <head> section. The <head> element contains metadata (document title, character set, styles, links, scripts), specific information about the web page that is not displayed to the user. The <meta> element is used to specify the metadata to provide browsers and search engines with technical information about the web page. For example, if you want to specify the author of your document, you may use the <meta> element like this:

```
<meta name="Author">
```

To define the character encoding for the document, you need to set the charset attribute with the "utf-8" value in nearly all cases. UTF-8 is the default character encoding for HTML5.

```
<meta charset="utf-8">
```

Use the <title> element to define the title of your document.

```
<title>HTML title</title>
```

Next is the <link> element which sets the relationship between the current document and the external resource. Generally, it is used to link to the external CSS stylesheet. Required attributes for the <link> element are rel, href and type.

```
<link rel="stylesheet" type="text/css" href="style.css">
```

Now, you can see the whole <head> section:

```
<head>
<title> HTML title.</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="Author">
<link rel="stylesheet" type="text/css" href="style.css">
</head>
```

The <body> element

The <body> of a document contains the content of the document. The content may be presented by a user agent in different ways. E.g., the content can be text, images, links, colors, graphics, etc.

```
<body>
```

```
  ...
</body>
```

Between the body tags, there can be different elements, to which you can give style by using CSS properties. Just add an id or class selector to your HTML element and in the <style> section mention your preferred styling options (color, size, font, etc.).

```
<style>
  .header-style {
    color: blue;
  }
</style>


<body>
<header class="header-style"></header>
</body>
```

The <script> element

In HTML5, the <script> tag is put to correspond the practices for embedding JavaScript. For example, it is related to the page loading speed.

```
<script src="js/scripts.js"></script>
```

3.2 CSS Reset

A CSS Reset (or "Reset CSS") is a short, often compressed (minified) set of CSS rules that resets the styling of all HTML elements to a consistent baseline. It 'reset' all of the default browser styles. We reset the browser styles for two primary reasons:

1. Not all browsers apply the same default rules. They may be similar, but not exact. It can be difficult to provide the same designs in each browser if the basic styles are different.
2. Once you start designing and coding all of the fine details of your site, you may discover that a lot of what you are doing is simply overriding default browser styles. The reset does this quickly so that you don't have to.

The Meyer's Reset Code

There are lots of different CSS Resets that we can use for our sites. You could even create your own. One of the two resets we'll use in this class is the Eric Meyer's Reset, created by CSS guru Eric Meyer.

Here's the code in the Meyer's Reset:

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
```

```
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
        margin: 0;
        padding: 0;
        border: 0;
        font-size: 100%;
        font: inherit;
        vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
        display: block;
}
body {
        line-height: 1;
}
ol, ul {
        list-style: none;
}
blockquote, q {
        quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
        content: '';
        content: none;
}
table {
        border-collapse: collapse;
        border-spacing: 0;
}
```

Including the Reset Stylesheet

We can include the reset style sheet using several different methods.

External Style Sheet:

We can use the reset style sheet as an external style sheet just like we do with our normal styles. Just make sure to add it first, since order matters.

```
<head>
<meta charset="utf-8"/>
<title>HTML Page for Testing CSS</title>
<link rel="stylesheet" href="reset.css" media="screen" />
<link rel="stylesheet" href="styles.css" media="screen" />
</head>
```

Remember, the CSS Reset style sheet should always go first.

Copy and Paste into Our Own Style Sheet:

You can also simply copy all of the rules from the reset style sheet and paste them into your own. Make sure that you put them at the top so that they don't override any of your rules. If you use this method, be sure to clearly mark the reset section of the style sheet and give credit to the author using CSS comments. While the author has made this style sheet available for everyone to use, this isn't your work, so don't pretend like it is.

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
        margin: 0;
        padding: 0;
        border: 0;
        font-size: 100%;
        font: inherit;
        vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
```

```
        display: block;
}
body {
        line-height: 1;
}
ol, ul {
        list-style: none;
}
blockquote, q {
        quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
        content: '';
        content: none;
}
table {
        border-collapse: collapse;
        border-spacing: 0;
}
/* End reset styles */

/* now we use our normal styles */
body { font-family: Georgia, serif; }
p { color: blue; }
h1, h2, h3, h4, h5, h6 { color: green; }
```

Sprinkled all throughout this, there were plenty of developers who went minimal by just zapping margin and padding from everything and leaving it at that:

```
* {
  padding: 0;
  margin: 0;
}
```

Normalize stylesheet

If the Meyer's reset seems a bit too much, there is a newer version which takes a slightly different approach: normalize.css. Instead of trying to eliminate the default browser styles, the normalize.css instead tries to normalize them to standard values across browsers. This removes some of the downsides of the Meyer's reset. It is also a newer and actively developed stylesheet, so it includes more modern HTML elements.

```
/*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */

/* Document

========================================================================
== */

/**
 * 1. Correct the line height in all browsers.
 * 2. Prevent adjustments of font size after orientation changes in iOS.
 */

html {
  line-height: 1.15; /* 1 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

/* Sections

========================================================================
== */

/**
 * Remove the margin in all browsers.
 */

body {
  margin: 0;
}

/**
 * Render the `main` element consistently in IE.
 */

main {
  display: block;
}

/**
 * Correct the font size and margin on `h1` elements within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
```

```
 */

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

/* Grouping content

========================================================================
== */

/**
 * 1. Add the correct box sizing in Firefox.
 * 2. Show the overflow in Edge and IE.
 */

hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}

/**
 * 1. Correct the inheritance and scaling of font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

pre {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/* Text-level semantics

========================================================================
== */

/**
 * Remove the gray background on active links in IE 10.
 */
```

```
a {
  background-color: transparent;
}

/**
 * 1. Remove the bottom border in Chrome 57-
 * 2. Add the correct text decoration in Chrome, Edge, IE, Opera, and Safari.
 */

abbr[title] {
  border-bottom: none; /* 1 */
  text-decoration: underline; /* 2 */
  text-decoration: underline dotted; /* 2 */
}

/**
 * Add the correct font weight in Chrome, Edge, and Safari.
 */

b,
strong {
  font-weight: bolder;
}

/**
 * 1. Correct the inheritance and scaling of font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

code,
kbd,
samp {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/**
 * Add the correct font size in all browsers.
 */
```

```
small {
  font-size: 80%;
}

/**
 * Prevent `sub` and `sup` elements from affecting the line height in
 * all browsers.
 */

sub,
sup {
  font-size: 75%;
  line-height: 0;
  position: relative;
  vertical-align: baseline;
}

sub {
  bottom: -0.25em;
}

sup {
  top: -0.5em;
}

/* Embedded content

========================================================================
== */

/**
 * Remove the border on images inside links in IE 10.
 */

img {
  border-style: none;
}

/* Forms
========================================================================
```

```
== */

/**
 * 1. Change the font styles in all browsers.
 * 2. Remove the margin in Firefox and Safari.
 */

button,
input,
optgroup,
select,
textarea {
  font-family: inherit; /* 1 */
  font-size: 100%; /* 1 */
  line-height: 1.15; /* 1 */
  margin: 0; /* 2 */
}

/**
 * Show the overflow in IE.
 * 1. Show the overflow in Edge.
 */

button,
input { /* 1 */
  overflow: visible;
}

/**
 * Remove the inheritance of text transform in Edge, Firefox, and IE.
 * 1. Remove the inheritance of text transform in Firefox.
 */

button,
select { /* 1 */
  text-transform: none;
}

/**
 * Correct the inability to style clickable types in iOS and Safari.
```

```
 */

button,
[type="button"],
[type="reset"],
[type="submit"] {
  -webkit-appearance: button;
}

/**
 * Remove the inner border and padding in Firefox.
 */

button::-moz-focus-inner,
[type="button"]::-moz-focus-inner,
[type="reset"]::-moz-focus-inner,
[type="submit"]::-moz-focus-inner {
  border-style: none;
  padding: 0;
}

/**
 * Restore the focus styles unset by the previous rule.
 */

button:-moz-focusring,
[type="button"]:-moz-focusring,
[type="reset"]:-moz-focusring,
[type="submit"]:-moz-focusring {
  outline: 1px dotted ButtonText;
}

/**
 * Correct the padding in Firefox.
 */

fieldset {
  padding: 0.35em 0.75em 0.625em;
}
```

```
/**
 * 1. Correct the text wrapping in Edge and IE.
 * 2. Correct the color inheritance from `fieldset` elements in IE.
 * 3. Remove the padding so developers are not caught out when they zero out
 *    `fieldset` elements in all browsers.
 */

legend {
  box-sizing: border-box; /* 1 */
  color: inherit; /* 2 */
  display: table; /* 1 */
  max-width: 100%; /* 1 */
  padding: 0; /* 3 */
  white-space: normal; /* 1 */
}

/**
 * Add the correct vertical alignment in Chrome, Firefox, and Opera.
 */

progress {
  vertical-align: baseline;
}

/**
 * Remove the default vertical scrollbar in IE 10+.
 */

textarea {
  overflow: auto;
}

/**
 * 1. Add the correct box sizing in IE 10.
 * 2. Remove the padding in IE 10.
 */

[type="checkbox"],
[type="radio"] {
  box-sizing: border-box; /* 1 */
```

```
  padding: 0; /* 2 */
}

/**
 * Correct the cursor style of increment and decrement buttons in Chrome.
 */

[type="number"]::-webkit-inner-spin-button,
[type="number"]::-webkit-outer-spin-button {
  height: auto;
}

/**
 * 1. Correct the odd appearance in Chrome and Safari.
 * 2. Correct the outline style in Safari.
 */

[type="search"] {
  -webkit-appearance: textfield; /* 1 */
  outline-offset: -2px; /* 2 */
}

/**
 * Remove the inner padding in Chrome and Safari on macOS.
 */

[type="search"]::-webkit-search-decoration {
  -webkit-appearance: none;
}

/**
 * 1. Correct the inability to style clickable types in iOS and Safari.
 * 2. Change font properties to `inherit` in Safari.
 */

::-webkit-file-upload-button {
  -webkit-appearance: button; /* 1 */
  font: inherit; /* 2 */
}
```

```
/* Interactive

========================================================================
== */

/*
 * Add the correct display in Edge, IE 10+, and Firefox.
 */

details {
  display: block;
}

/*
 * Add the correct display in all browsers.
 */

summary {
  display: list-item;
}

/* Misc

========================================================================
== */

/**
 * Add the correct display in IE 10+.
 */

template {
  display: none;
}

/**
 * Add the correct display in IE 10.
 */

[hidden] {
  display: none;
}
```

Summary

1. HTML is used to design web pages using markup elements or tags. HTML5 is the fifth version of HTML and includes a number of tags with new functions for formatting and displaying content.
2. The goal of a reset stylesheet is to reduce browser inconsistencies in things like default line heights, margins and font sizes of headings, and so on.

Keywords

- HTML page structure
- CSS reset
- Meyer's Reset Code

Self Assessment

1. HTML document has three main elements. Which of the following is not one of them?

A. DTD

B. Head

C. Neck

D. Body

2. In which part of HTML document we can define meta data?

A. <!DOCTYPE>

B. <HTML>

C. <HEAD>

D. <BODY>

3. Which of the following tag can't go inside <head> element?

A. Headings

B. Titles

C. Meta Data

D. Scripts

4. The correct sequence of HTML tags for starting a webpage is -

A. Head, Title, HTML, body

B. HTML, Body, Title, Head

C. HTML, Title, Head, Body

D. HTML, Head, Title, Body

5. ___ contains content of the HTML document

A. Html

B. Body

C. Title

D. Head

6. Which tag is used to embed JavaScript?

A. <script>

B. <scripting>

7. <link> element is set in which of the following elements?

A. <title>

B. <html>
C. <body>
D. <head>

8. _____ resets the styling of html elements.
A. CSS set
B. CSS reset
C. CSS preset
D. None

9. Which of the following is true?
A. CSS reset can included as only internal stylesheet
B. CSS reset can included as only external stylesheet
C. CSS reset can included as both external and internal stylesheet
D. None of these is true.

10. CSS reset is also called as ___
A. Mayor's Reset
B. Meyer's Reset
C. Major Reset
D. None of the above

11. HTML is stand for _____
A. Hyper Text Markup Language
B. Holistick Technical Method Library
C. Hyper Tax Makes Line
D. None of the above

12. The BODY tag is usually used after _____.
A. HTML tag
B. EM tag
C. TITLE tag
D. HEAD tag

13. HTML is the standard ____language for creating Web pages.
A. scripting
B. programming
C. styling
D. markup

14. Which tag is the root element of an HTML page?
A. <html>
B. <title>
C. <head>
D. <body>

15. All headings should be used within the header element.
A. True
B. False

Answers for Self Assessment

1.	C	2.	C	3.	A	4.	D	5.	B
6.	A	7.	D	8.	B	9.	C	10.	B
11.	A	12.	D	13.	A	14.	A	15.	B

Review Questions

1. What is the purpose of Doctype in an html document?
2. Explain the importance of head section in an html document?
3. What are the different parts of an HTML5 document? Explain with the help of an example
4. Illustrate the significance of CSS reset.
5. Differentiate between Meyer's Reset and normalize stylesheet.
6. Illustrate the concept of Meyer's Reset using external stylesheet.
7. Illustrate Normalize stylesheet with example.
8. Why is CSS reset used? Demonstrate.
9. Differentiate between head and body section of your HTML document.
10. How can we define the link of stylesheet in our HTML document.

Further Readings

https://www.w3.org/TR/html4/struct/global.html#:~:text=An%20HTML%204%20document%20is,contains%20the%20document's%20actual%20content.

https://piccalil.li/blog/a-modern-css-reset/

Unit 04:HTML5 Structure for Website

CONTENTS

Objectives

After studying, you will be able to:

- Understand and implement the <nav> element in HTML
- Understand and implement 4-column and 2-column layout in CSS.

Introduction

We will learn about navigation section element and column layouts in HTML and CSS. On a website, a navigation menu is an organized list of links to other web pages, usually internal pages. Navigation menus appear most commonly in page headers or sidebars across a website, allowing visitors to quickly access the most useful pages. Navigation does more than help us move from one web page to another — it also helps us understand the relationships between individual pages on a website.In other words, the website's IA isn't visible in the navigation interface, but it is the foundation of that interface. This gives visitors the sense that the content is connected and categorized to meet their needs and expectations — without actually showing all the spreadsheets and diagrams that went into identifying and organizing those relationships among your content.

Web development is a playground of innovation and when it appears a new worthy feature, it usually grows into the soil firmly. It also happened with multi column layout, too. It was adopted to web development once and now we see how often it is used by modern designers and how fast it is developing.

4.1 <nav>: The Navigation Section Element

The <nav> HTML element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents. Common examples of navigation sections are menus, tables of contents, and indexes.In this example, a <nav> block is used to contain an ordered list (<ol>) of links. With appropriate CSS, this can be presented as a sidebar, navigation bar, or drop-down menu.

```
<!DOCTYPE html>
<htmllang="en">
<head>
```

```
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
nav {
   border-bottom: 1px solid black;
}
.navigation ol {
   list-style-type: none;
   padding: 0;
}
.item {
   display: inline-block;
}
</style>
</head>
<body>
<nav class="navigation">
<ol>
<li class="item"><a href="#">Home</a></li>
<li class="item"><a href="#">Products</a></li>
<li class="item"><a href="#">About us</a></li>
</ol>
</nav>
</body>
</html>
```

The output of the above code is shown in Figure 1.

Figure 1

If in the above example, display: inline-block property is not used, it leads to a vertical navigation bar

```
<!DOCTYPE html>
<htmllang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
nav {
   border-bottom: 1px solid black;
}
.navigation ol {
   list-style-type: none;
   padding: 0;
}
/* .item {
   display: inline-block;
} */
</style>
</head>
<body>
<nav class="navigation">
<ol>
<li class="item"><a href="#">Home</a></li>
<li class="item"><a href="#">Products</a></li>
<li class="item"><a href="#">About us</a></li>
</ol>
</nav>
</body>
</html>
```

The output looks as in Figure 2.

Figure 2

The semantics of the nav element is that of providing links. However a nav element doesn't have to contain a list, it can contain other kinds of content as well. In this navigation block, links are provided in prose:

```
<nav>
<h2>Navigation</h2>
<p>
    You are on my home page. To the north lies <a href="/blog">my blog</a>, from
    whence the sounds of battle can be heard. To the east you can see a large
    mountain, upon which many <a href="/school">school papers</a> are littered.
    Far up thus mountain you can spy a little figure who appears to be me,
    desperately scribbling a <a href="/school/thesis">thesis</a>.
</p>
<p>
    To the west are several exits. One fun-looking exit is labeled
<a href="https://games.example.com/">"games"</a>. Another more
    boring-looking exit is labeled<a href="https://isp.example.net/">ISP™</a>.
</p>
<p>
    To the south lies a dark and dank <a href="/about">contacts page</a>.
    Cobwebs cover its disused entrance, and at one point you see a rat run
    quickly out of the page.
</p>
</nav>
```

The output of above code is shown in Figure 3.

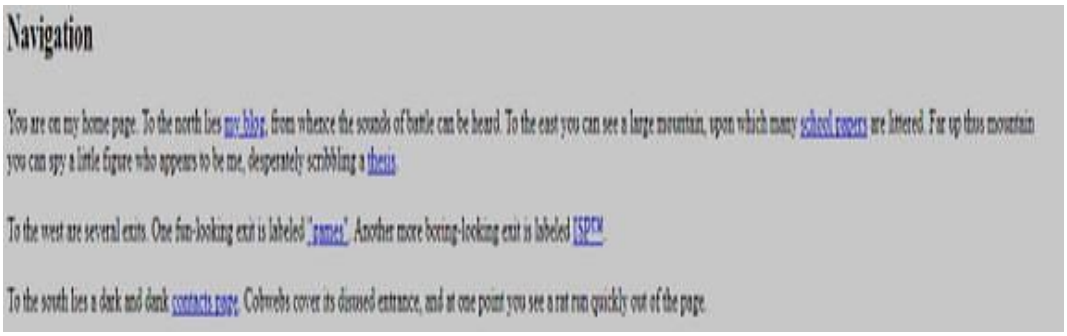


Figure 3

4.2 Create Columns in HTML

There is more than one way to create HTML columns; despite how it sounds, you need a combination of HTML and CSS. For a long time, the only way to create columns was to manually declare them by setting the CSS styles for the page with static sizes for items. This approach worked, but it only simulated the visual appearance of columns.

Since then, CSS has grown, and building columned layouts has become much more accessible and has much more support than it used to. Moreover, there are many ways to create column layouts for web pages.

4-column layout

To create a 4-column layout grid with CSS, the code is as follows –

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style>
  * {
    box-sizing: border-box;
  }
  .first, .second, .third, .fourth {
    float: left;
    width: 25%;
    color: white;
    padding: 10px;
    height: 500px;
    text-align: center;
  }
  .first {
    background-color: tomato;
  }
```

```
  .second {
    background-color: teal;
  }
  .third {
    background-color: rgb(166, 71, 255);
  }
  .fourth {
    background-color: rgb(255, 71, 194);
  }
  .container:after {
    clear: both;
  }
</style>
</head>
<body>
<h1 style="text-align: center;">Four Column grid example</h1>
<div class="container">
<div class="first">
<h1>Some text on the first div</h1>
</div>
<div class="second">
<h1>Some text on the second div</h1>
</div>
<div class="third">
<h1>Some text on the third div</h1>
</div>
<div class="fourth">
<h1>Some text on the fourth div</h1>
</div>
</div>
</body>
</html>
```

The output of above code is shown in Figure 4.

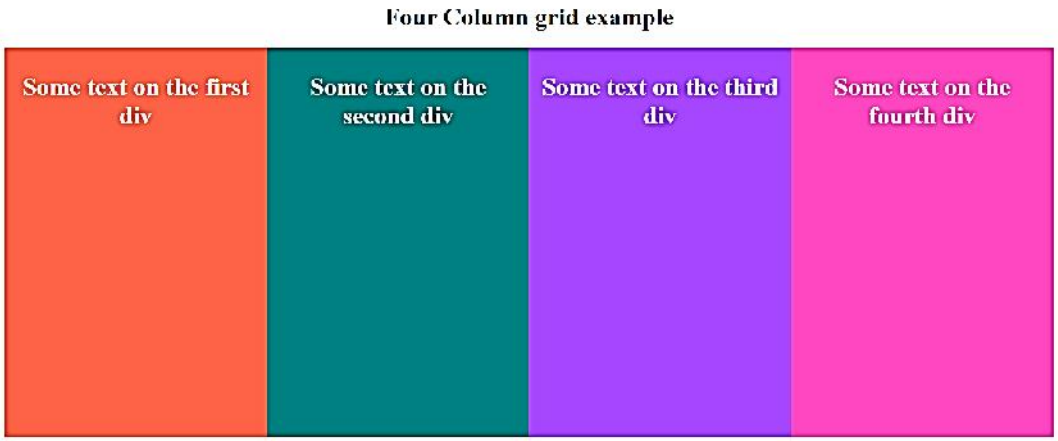


Figure 4

2-column layout

To create a 2-column layout grid with CSS, the code is as follows –

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style>
  .left, .right {
    height: 50%;
    width: 50%;
    position: fixed;
    overflow-x: hidden;
    padding-top: 20px;
  }
  .left {
    left: 0;
    background-color: rgb(36, 0, 95);
  }
  .right {
    right: 0;
    background-color: rgb(56, 1, 44);
  }
</style>
</head>
<body>
<h1 style="color: black;">Two column layout grid example</h1>
```

```
<div class="left">
<h1>Some random text on the left</h1>
</div>
<div class="right">
<h1>Some random text on the right</h1>
</div>
</body>
</html>
```

The output of above code is shown in Figure 5.

Figure 5

4.3 CSS Multi-Column Layout

The multiple-column layout specification provides you with a method for laying content out in columns, as you might see in a newspaper.CSS Multi-column Layout is a module of CSS that adds support for multi-column layouts. Support is included for establishing the number of columns in a layout, as well as how content should flow from column to column, gap sizes between columns, and column dividing lines (known as column rules) along with their appearance.

In the following example, the column-count property has been applied to the element with the class container. As the value of column-count is 4, the content is arranged into four columns of the same size.

```
<!DOCTYPE html>
<htmllang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Multi-column</title>
<style>
.container {
```

```
  column-count: 4;
}
.first, .second, .third, .fourth {
    color: white;
    padding: 10px;
    height: 500px;
    text-align: center;
  }
.first {
    background-color: tomato;
  }
  .second {
    background-color: teal;
  }
  .third {
    background-color: rgb(166, 71, 255);
  }
  .fourth {
    background-color: rgb(255, 71, 194);
  }
</style>
</head>
<body>
<div class="container">
<div class="first">
<h1>Some text on the first div</h1>
</div>
<div class="second">
<h1>Some text on the second div</h1>
</div>
<div class="third">
<h1>Some text on the third div</h1>
</div>
<div class="fourth">
<h1>Some text on the fourth div</h1>
</div>
</div>
</body>
</html>
```

The output of above example is shown in Figure 6.

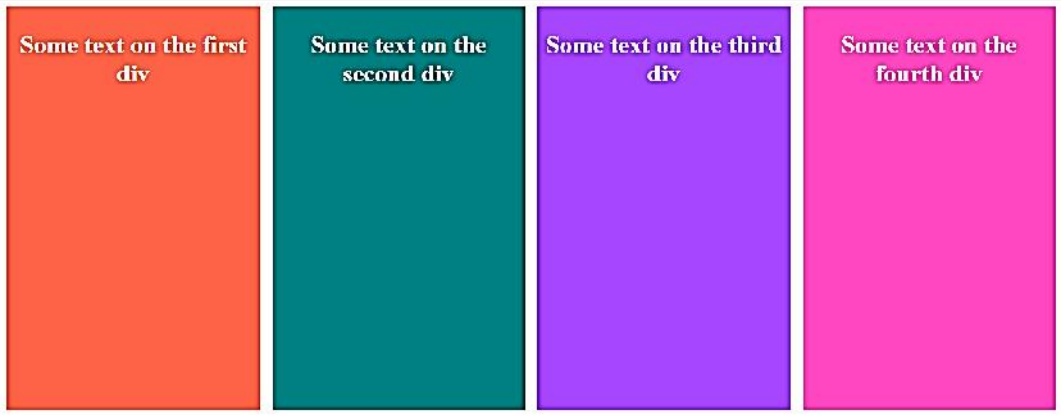


Figure 6

Similarly, 2-column or 3-column layout can also be created using column-count property in CSS.

You may also modify the above code to include column-gap and column-rule properties. column-gap is used to specify the gap between the columns while column-ruleproperty sets the width, style, and color of the rule between the columns of the element

```
<!DOCTYPE html>
<htmllang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Multi-column</title>
<style>
.container {
 column-count: 4;
 column-gap: 70px;
 column-rule: 5px solid black;
}
.first, .second, .third, .fourth {
    color: white;
    padding: 10px;
    height: 500px;
    text-align: center;
  }
.first {
    background-color: tomato;
  }
```

```
  .second {
    background-color: teal;
  }
  .third {
    background-color: rgb(166, 71, 255);
  }
  .fourth {
    background-color: rgb(255, 71, 194);
  }
</style>
</head>
<body>
<div class="container">
<div class="first">
<h1>Some text on the first div</h1>
</div>
<div class="second">
<h1>Some text on the second div</h1>
</div>
<div class="third">
<h1>Some text on the third div</h1>
</div>
<div class="fourth">
<h1>Some text on the fourth div</h1>
</div>
</div>
</body>
</html>
```

The output of above example is shown in Figure 7.

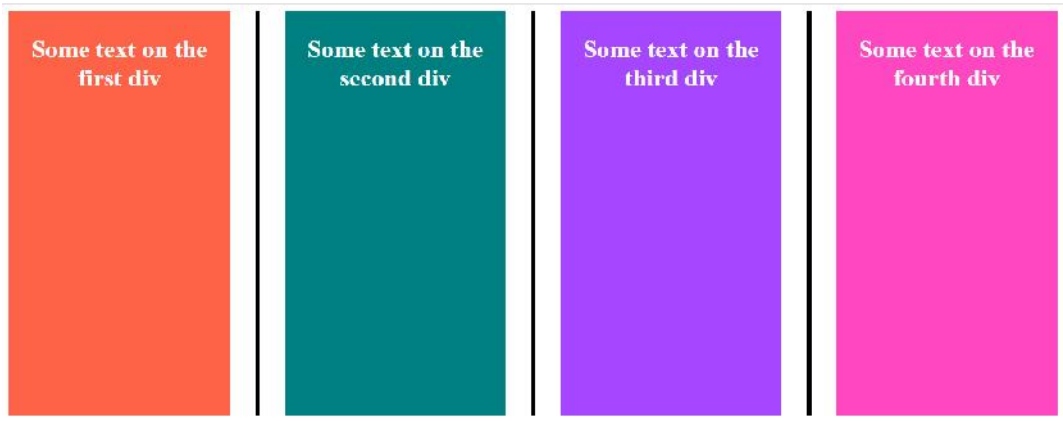


Figure 7

If you are not using column-rule shorthand property, then column-rule-color, column-rule-style and column-rule-width can also be used for setting the color, style of border and width of border respectively.

Summary

1. The <nav> tag is one of the HTML5 elements. It is used to specify a block of navigation links, either within the current document or to other documents.
2. In web design, columns are often used to separate primary content from secondary and tertiary content. For example, a common two column layout may include a left column with navigation links, and a right column for body text.

Keywords

- Navigation
- Column-layout
- 2-column layout
- 4-column layout

Self Assessment

1. What is a navigation bar?

A. Website page sourcing

B. Open tags for a site

C. List of links

D. A2 + b2 = c2

2. The ___ tag defines a set of navigation links

A. <navigation>

B. <navigator>

C. <navigate>

D. <nav>

3. Navigation bars can only be created vertically.

A. True

B. False

4. A navigation bar needs standard HTML as a base.

A. True

B. False

5. What is the purpose of following code in a navigation bar?

```
{
display: inline-block
}
```

A. Creates a vertical navigation bar

B. Creates a diagonal navigation bar

C. Creates a horizontal navigation bar

D. None of the above

6. What will be the result if the following code is not used in a navigation bar?

```
{
display: inline-block
}
```

a. Creates a diagonal navigation bar
b. Creates a horizontal navigation bar
c. None of these
d. Creates a vertical navigation bar

7. Which tag is used for creating individual menus in a navigation bar?

A. <p>

B.

C. <list>

D. None of the above

8. <nav> can only contain list and nothing else

A. False

B. True

9. Which of the following property defines the number of columns in a multicolumn text flow?

A. columns

B. column-flow

C. column-count

D. column-number

10. Which of the following property defines the gap between columns in a multicolumn text flow?

A. column-rule-flow

B. column-rule

C. none of the mentioned

D. column-gap

11. This shorthand CSS property sets the width, style, and color of the line drawn between columns in a multi-column layout.

A. column-color

B. column-rule

C. column-rule-style

D. column-rule-decoration

12. Which of the following property defines the width of a rule between columns in a multicolumn text flow?

A. column-rule-width

B. column-rule-style

C. column-width

D. columns

13. Which of the following property defines the width of a rule between columns in a multicolumn text flow?

A. column-rule-width

B. column-width

C. column-rule-style

D. columns

14. Creating columns in CSS is not possible without using column properties.

A. True

B. False

15. Which of the following property defines the style of the divider rule between columns in a multicolumn text flow?

A. columns

B. column-flow

C. column-style

D. none of the mentioned

Answers for Self Assessment

l.	C	2.	D	3.	B	4.	A	5.	C
6.	D	7.	B	8.	A	9.	C	10.	D
11.	B	12.	A	13.	C	14.	B	15.	D

Review Questions

1. Define the concept of navigation in websites.
2. Which html element is used to create a navigation bar? Discuss.
3. Apply the concept of <nav> element and create a horizontal navigation bar.
4. Apply the concept of <nav> element and create a vertical navigation bar.
5. Describe how can we create columns in HTML.
6. Create a 3-column layout using the concept of multicolumn layout.
7. Create a 5-column layout without using the concept of multicolumn layout.

8. Is navigation required to comprise only list? Discuss.
9. Why is there a need to create multiple columns in a website? Explain with example.
10. Compare and contrast different column properties of a multi-column layout with examples.

Further Readings

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Multiple-column_Layout

Unit 5: Using CSS

CONTENTS

Objectives

1. Understand the concept of navigation bars
2. Implement navigation bars with variations

Introduction

GUI includes a navigation system or bar that facilitates information access for users. Links to the various parts of a website are found in the user interface element of a webpage. Most often, a navigation bar appears at the top of the page as a horizontal list of links. It can be positioned either before or below the header, but it must always come before the page's main content. The navigation of a website must be simple to use. It has a significant impact on the website since it enables easy access to any part for users.

5.1 Writing CSS for navigation bar

Horizontal navigation bar

The list of links that runs horizontally across the top of the page is known as the horizontal navigation bar. Let's use an example to demonstrate how to make a horizontal navigation bar.

In this illustration, we include the overflow: hidden property to stop the li elements from leaving the list. display: block property displays the links as the block elements and makes the entire link area clickable. Additionally, we are adding the float: left attribute, which makes use of float to have the block elements move next to one another. We must apply the background-color attribute to the <ul> element rather than the <a> element if we want the full-width background color.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
 list-style-type: none;
```

```
  margin: 0;
  padding: 0px;
  overflow: hidden;
  background-color: lightgreen;
}
li {
  float: left;
}
li a {
  display: block;
color: black;
 font-size:20px;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none;
}
.active{
background-color: darkgreen;
color: white;
}
li a:hover {
  background-color: yellow;
}
</style>
</head>
<body>
<ul>
<li><a class="active" href="#home">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Products</a></li>
<li><a href="#">Contact</a></li>
</ul>
</body>
</html>
```

The output of above example is shown in Figure 1.

Figure 1

Border dividers

Using the border-right property, we can place a border between the links in the navigation bar. The following illustration clarifies it better.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0px;
  overflow: hidden;
  background-color: lightgreen;
}
li {
  float: left;
    border-right: 1px solid darkgreen;
}
li a {
  display: block;
color: black;
 font-size:20px;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none;
}
.active{
background-color: darkgreen;
```

```
color: white;
}
li a:hover {
  background-color: yellow;
}
</style>
</head>
<body>
<ul>
<li><a class="active" href="#home">Home</a></li>
<li><a href="#">Java</a></li>
<li><a href="#">HTML</a></li>
<li><a href="#">CSS</a></li>
</ul>
</body>
</html>
```

The output of above example is shown in Figure 2.

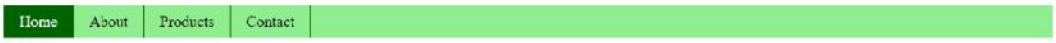


Figure 2

Fixed Navigation bars

Fixed navigation bars remain at the bottom or top of the page even when we scroll. View an illustration of the same.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  position: fixed;
```

```
  width:100%;
top:0;
  margin: 0;
  padding: 0px;
  overflow: hidden;
  background-color: lightgreen;
}
li {
  float: left;
    border-right: 1px solid darkgreen;
}
li a {
  display: block;
color: black;
 font-size:20px;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none;
}
.active{
background-color: darkgreen;
color: white;
}
li a:hover {
  background-color: yellow;
color: white;
}
</style>
</head>
<body>
<ul>
<li><a class="active" href="#home">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Products</a></li>
<li><a href="#">Contact</a></li>
</ul>
<h2 style="padding-top: 200px; text-align: center;">Scroll down the page to see the fixed navigation bar</h2>
</body>
</html>
```

The output of above example is shown in Figure 3.

Figure 3

In the above image, taking into consideration the position of the scroll bar, it is clear that the navigation bar is fixed

Sticky Navbar

The element is positioned according to the user's scroll position using the position: sticky; property. When the scroll reaches a specific position, the items might stick thanks to this CSS feature. A sticky element switches between fixed and relative property based on the scroll position.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  position: sticky;
  width:100%;
top:0;
  margin: 0;
  padding: 0px;
  overflow: hidden;
  background-color: lightgreen;
}
li {
  float: left;
    border-right: 1px solid darkgreen;
}
li a {
  display: block;
color: black;
```

```
 font-size:20px;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none;
}
.active{
background-color: darkgreen;
color: white;
}
li a:hover {
  background-color: yellow;
}
</style>
</head>
<body>
<h1> Example of sticky navigation bar</h1>
<ul>
<li><a class="active" href="#home">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Products</a></li>
<li><a href="#">Contact</a></li>
</ul>
<h2 style="padding-top: 200px; padding-bottom:1000px;text-align: center;">Scroll down the page to see the sticky navigation bar</h2>
</body>
</html>
```

The output of above example is shown in Figure 5 and Figure 6

Figure 5

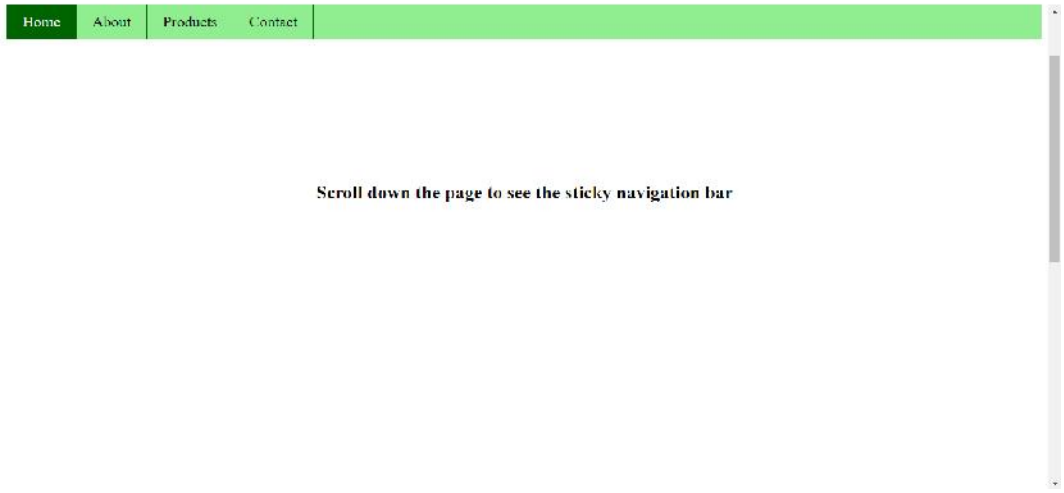


Figure 6

Dropdown Navbar

Following example explain how to create a dropdown menu inside a navigation bar.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: lightgreen;
}
li {
  float: left;
}
li a, .dropbtn {
  display: inline-block;
color: black;
 font-size:20px;
  text-align: center;
  padding: 10px 20px;
  text-decoration: none;
}
.active{
background-color: darkgreen;
```

```
color: white;
}
li a:hover , .dropdown:hover .dropbtn{
  background-color: orange;
color: white;
}
.dropdown-content {
  display: none;
  position: absolute;
  background-color: lightblue;
  min-width: 160px;
  box-shadow: 5px 8px 10px 0px black;
 }
.dropdown-content a {
color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
  text-align: left;
}
.dropdown-content a:hover {
background-color: gray;
color:white;
}
.dropdown:hover .dropdown-content {
  display: block;
}
h1,h2,h3{
text-align:center;
color: green;
}
</style>
</head>
<body>
<ul>
<li><a class="active" href="#home">Home</a></li>
<li><a href="#">About</a></li>
<li class="dropdown">
<a href="#" class="dropbtn">Products</a>
<div class="dropdown-content">
```

```
<a href="#">Men</a>
<a href="#">Women</a>
<a href="#">Kids</a>
</div>
</li>
<li><a href="#">Contact</a></li>
</ul>
<h3>To view the dropdown effect, move your mouse over the "Products" section</h3>
</body>
</html>
```

The output of above example is shown in Figure 7.

Figure 7

Vertical Navigation bar

In this example, we are going to see how to build a vertical navigation bar.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 200px;
  background-color: lightgreen;
}
li a {
  display: block;
```

```
color: black;
  font-size:20px;
  padding: 8px 16px;
  text-decoration: none;
}
.active{
  background-color: darkgreen;
color: white;
}
li a:hover {
  background-color: yellow;
}
</style>
</head>
<body>
<h2>Vertical Navigation Bar</h2>
<ul>
<li><a href="#" class = "active">Home</a></li>
<li><a href = "#">About</a></li>
<li><a href = "#">Products</a></li>
<li><a href = "#">Services</a></li>
<li><a href = "#">Contact</a></li>
</ul>
</body>
</html>
```

The output of above example is shown in Figure 8.

Figure 8

Full-height fixed Vertical Navbar

Use the parameters height: 100%; and position: fixed to create a fixed full-height side navigation bar.

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
background-color: rgb(184, 250, 184)
}
ul {
list-style-type: none;
margin: 0;
padding: 0;
height:100%;
top:0;
width:150px;
overflow: auto;
background-color: lightgreen;
border: 1px solid darkgreen;
position: fixed;
}
li a {
display: block;
color: black;
font-size:20px;
padding: 10px 20px;
text-decoration: none;
border-bottom: 1px solid black;
}
.active{
background-color: darkgreen;
color: white;
}
li a:hover {
background-color: yellow;
}
</style>
</head>
<body>
```

```
<ul>
<li><a href = "#" class = "active">Home</a></li>
<li><a href = "#">About</a></li>
<li><a href = "#">Product</a></li>
<li><a href = "#">Services</a></li>
<li><a href = "#">Contact</a></li>
</ul>
<div style="margin-left:20%;padding-bottom:2000px;">
<h2>Side navigation bar with height: 100%; and position: fixed;</h2>
<h3>Scroll the page, and see how the sidenav sticks to the page</h3>
</div>
</body>
</html>
```

The output of above example is shown in Figure 9.

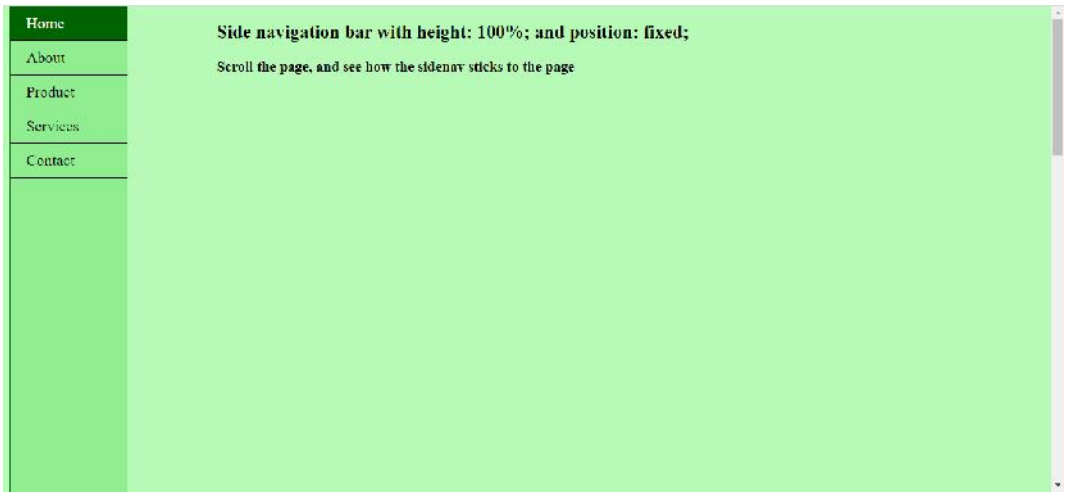


Figure 9

Summary

1. A navigation bar along X axis is a horizontal navbar.
2. To create a horizontal navbar, float: left can be used for block elements to float next to each other
3. border-right property can be used to separate the links in a horizontal navigation bar
4. position: fixed is used for a fixed navigation bar
5. position: sticky is used for a sticky navigation bar
6. The difference between position fixed vs sticky is that fixed always positions an element relative to the viewport, while sticky behaves like a regular element until it reaches the defined offset and then becomes fixed.
7. height: 100% allows us to create a fixed full-height side navigation bar.

Keywords

- Horizontal navbar
- Vertical navbar

Self Assessment

1. Navigation bar can be __
   A. Vertical
   B. Horizontal
   C. Both
   D. None
2. float: left is used to align list items in __
   A. Vertical navbar
   B. Horizontal navbar
3. Which of the following makes use of float to have the block elements move next to one another?
   A. float: left
   B. text-align: center;
   C. display: block;
   D. None
4. To place a border between the links in the navigation bar, we can use __ property
   A. margin
   B. padding
   C. none
   D. border
5. To create a fixed navigation bar, we use ___.
   A. display: fixed
   B. float: fixed
   C. position: fixed
   D. none
6. To create a sticky navigation bar, we use ___.
   A. display: sticky
   B. position: sticky
   C. float: sticky
   D. none
7. In a _____ bar, we have the list items placed next to each other
   A. Horizontal navigation
   B. Vertical navigation
   C. Both
   D. None
8. An element with ___ will behave like a relatively-positioned element until it reaches a specified point and then starts behaving like a statically-positioned element.
   A. position: relative
   B. position: fixed
   C. none
   D. position: sticky
9. In a dropdown menu, why do we use display: none?
   A. We don't want to display at all
   B. Because there are already so many items
   C. Initially we just want it hidden
   D. None
10. _____ is a navigation menu component stretching along the side of a page to present all links that will take users to different pages or parts of a website or mobile app.
    A. Horizontal navbar
    B. Vertical navbar
11. In order to create a full-height navbar, we use ___ in the ordered list
    A. display: 100%
    B. display: 100px
    C. display: 100em
    D. display: 100rem
12. In order to provide a full-width background color to the navigation bar, we must apply background-color property to ___ element.

A. <a>
B.
C. none
D.

13. A ___ navbaralways positions an element relative to the viewport
    A. Sticky
    B. Both a and c
    C. Fixed
    D. None
14. A ____ navbarbehaves like a regular element until it reaches the defined offset and then becomes fixed.
    A. Fixed
    B. Sticky
    C. Both
    D. None
15. While working with position sticky for a sticky navigation bar, we must take care:
    A. Is offset specified (CSS top property)
    B. Is there an overflow property set for the parent element
    C. Is there a height property set for the parent element
    D. All of the above

Answers for Self Assessment

l.	C	2.	B	3.	A	4.	D	5.	C
6.	B	7.	A	8.	D	9.	C	10.	B
11.	A	12.	D	13.	C	14.	B	15.	D

Review Questions

1. Illustrate the importance of a navigation bar in websites.
2. Demonstrate the concept of a horizontal navigation bar with links separated by borders.
3. Design a fixed navigation bar with the following as links:
4. Missions
5. Galleries
6. NASA TV
7. Follow NASA
8. Downloads
9. About
10. NASA Audiences
11. Design a sticky navigation bar with the following as links:
12. Home
13. Students
14. Faculty

15. Staff
16. Media
17. Alumni
18. Industry
19. Resources
20. Explain the purpose of using float:left while creating a horizontal navigation bar.
21. Differentiate between position: fixed and position: sticky with the help of an example.
22. Create a vertical navigation bar with the following a links:
23. Top offers
24. Grocery
25. Mobiles
26. Fashion
27. Electronics
28. Appliances
29. Travel
30. Describe the concept of a full height navigation bar with the help of an example.
31. Illustrate the concept of a dropdown navigation
32. Demonstrate the importance of using display:block while creating a navigation bar in CSS.

Further Readings

https://www.w3schools.com/tags/tag_header.asp

https://www.w3schools.com/tags/tag_footer.asp

Unit 6:Using CSS

CONTENTS

Objectives

1. Understand and implement header and footer elements
2. Understand and implement the concept of formatting images in columns

Introduction

HTML tags are (primarily) used to format material; they instruct the browser how to present the content on the page. If you are familiar with HTML in any way, you will be aware of this. They don't specify the kind of material they include or the function that content serves on the page.By creating particular tags that make it obvious what function the content those tags contain plays, Semantic HTML5 tackles this flaw.By including semantic HTML tags on your sites, you can provide Google and Bing more details about the functions and relative relevance of the various elements on your website.By using Semantic tags in our code, we can provide additional information about that document by defining the layout and sections of the webpage.Header and footer are among the semantic elements in HTML. The material that should be regarded as the introduction to a page or section is defined by the "header" element.Footer is used at the bottom of a page or section. Contact details and basic site navigation may be included. Various Semantic tags are:

Header element – The <header> element defines content that should be considered the introductory information of a page or section.

Nav element – Main navigation menu links would all be placed in a <nav> tag. But sub navigation menus elsewhere on the page could also get one.

Main tag – The body of a page should go in the <main> tag – not sidebars and main navigation. There should be only one per page.

Article element – The <article> element defines self-contained content that could stand independently of the page or site it's on. For example, a blog post.

Section element – Using <section> is a way of grouping together nearby content of a similar theme. A section tag differs to an article tag because it isn't necessarily self-contained.

Aside element – An <aside> element defines content that's less important. It's often used for sidebars – areas that add complementary but not vital information.

Footer element – You would use <footer> at the base of a page or section. It might include contact information and some site navigation.

6.1 Header

The <header>elememt designates introductory information, which is often a collection of introductions or navigational tools. It could also have a logo, a search form, the author's name, and other features in addition to certain header components. We can understand this basic header with the help of following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
header.page-header {
background-image:url(1.jpg);
   background-repeat: no-repeat;
   background-size: cover;
   display: flex;
   height: 150px;
   align-items: center;
color: #fff;
}
</style>
</head>
<body>
<header class="page-header">
<h1>This is a header</h1>
</header>
</body>
</html>
```

The output of above example is shown in Figure 1.

Figure 1

Initial usage of <header>

The <header> element was first used for headings in the very first version of HTML. At some stage, headings evolved into <h1> through <h6>, freeing up <header> to perform a different function.

Another usage

With the exception of when it is nested inside sectioning content, the <header> element has the same meaning as the site-wide banner landmark function. The header element is not a landmark in such case. We can understand this with the help of following examples.

Page Header example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<header>
<h1>Main Page Title</h1>
<imgsrc="1.jpg">
</header>
</body>
</html>
```

The output of above example is shown in Figure 2.

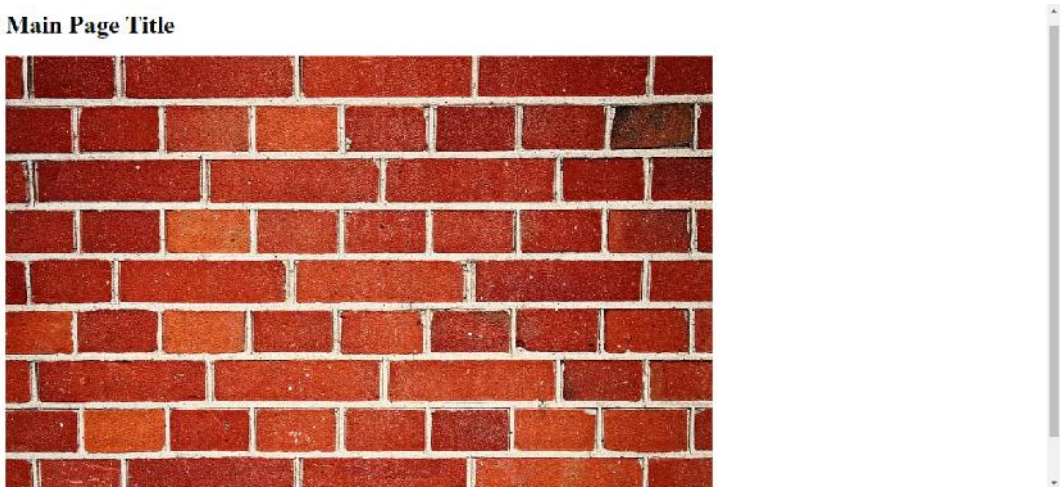


Figure 2

Article header example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<article>
<header>
<h2>The Planet Earth</h2>
<p>
    Posted on Wednesday, <time datetime="2017-10-04">4 October 2017</time> by
    Jane Smith
</p>
</header>
<p>
   We live on a planet that's blue and green, with so many things still unseen.
</p>
<p><a href="https://example.com/the-planet-earth/">Continue reading…</a></p>
</article>
</body>
</html>
```

The output of above example is shown in Figure 3.

Figure 3

6.2 Footer

Footer represents data related to the author, data that contains copyright information or some links related to that data. This author information enclosed within <address> element. This type of tag known as <footer> tag in HTML. Footer Tag in HTML is defined within <footer>tag.One can use footer element within <article>, <body>, <aside>, <section>, <nav>, <details>, <fieldset>, or on the <figure>. It is always placed within the <body> tag. It is included within the individual blocks. Footer Tag in HTML used to show the footer part of the webpage or any section. <footer> doesn't work as sectioning element so it doesn't create a new section in the outline.

Syntax for <footer> tag:

```
<body>
<footer>
<p>Some text</p>
</footer>
</body>
```

Following is the example uses footer with CSS:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
footer {
   display: flex;
   justify-content: center;
   padding: 10px;
```

```
    background-color: seagreen;
color: white;
}
</style>
</head>
<body>
<article>
<h1>Companies visited</h1>
<ol>
<li>Amazon.com, Inc.</li>
<li>Wipro Limited</li>
<li>Amdocs</li>
<li>Microsoft Corporation</li>
</ol>
<footer>
<p>Copyright © 2022, ABC Institute of Engineering and Technology</p>
</footer>
</article>
</body>
</html>
```

The output of above example is shown in Figure 4.

Figure 4

6.3 Example using header, footer and other semantic elements

Consider the following example:

```
<!Doctype Html>
<html>
```

```
<head>
<title></title>
</head>
<body bgcolor=lightyellow>
<header>
<h1>HTML5 includes new semantics</h1>
<p>It includes semantic tags like header,footer, nav</p>
</header>
<header>
<h1>Example of complete HTML5 Basics</h1>
<h2>The markup of the future under development.</h2>
</header>
<nav>
<h1><p>The nav element represents a section of navigation links. It is suitable for either site navigation or a table of contents. </p></h1>
<a href="/">http://www.gmail.com</a><br>
<a href="http://www.facebook.com">Facebook website</a><br>
</nav>
<aside>
<h1>Other education based websites of State</h1>
<a href="http://mahahsscboard.ac.in">State Board website</a><br>
<a href="http://examinfo.mhhsc.ac.in">Online Exam Website<a/><br>
</aside>
<section>
<h1>Impressive Web Designing</h1>
<p>The aside element is for content that is tangentially related to the content around it, and is typically useful for marking up sidebars. </p></section><section>
<h1>Articles on:Article tag</h1>
</section>
<article>
<p>The article element represents an independent section of a document, page or site. It is suitable for content like news or blog articles, forum posts or individual comments.
</p>
</article>
<footer>© 2022 ABC</footer>
</body>
</html>
```

The output of above example is shown in Figure 5.

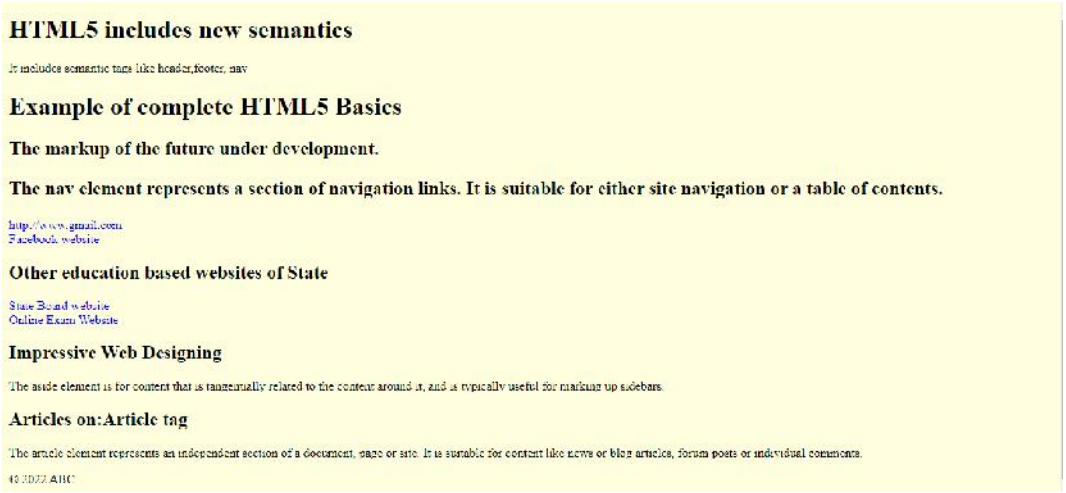


Figure 5

6.4 Formatting images in columns

Following code explains this example

```
<!DOCTYPE html>
<html>
<style>
* {
  box-sizing: border-box;
}
body {
  margin: 0;
  font-family: Arial;
}
.header {
  text-align: center;
  padding: 32px;
}
/* Create two equal columns that floats next to each other */
.column {
  float: left;
  width: 50%;
  padding: 10px;
}
.column img {
  margin-top: 12px;
}
/* Clear floats after the columns */
```

```
.row:after {
  content: "";
  display: table;
  clear: both;
}
</style>
<body>

<!-- Header -->
<div class="header">
<h1>Image Grid</h1>
</div>

<!-- Photo Grid -->
<div class="row">
<div class="column">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
</div>
<div class="column">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
</div>
<div class="column">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
```

```
<imgsrc="1.jpg" style="width:100%">
</div>
<div class="column">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
<imgsrc="1.jpg" style="width:100%">
</div>
</div>
</body>
</html>
```

The output of above example is shown in Figure 6.

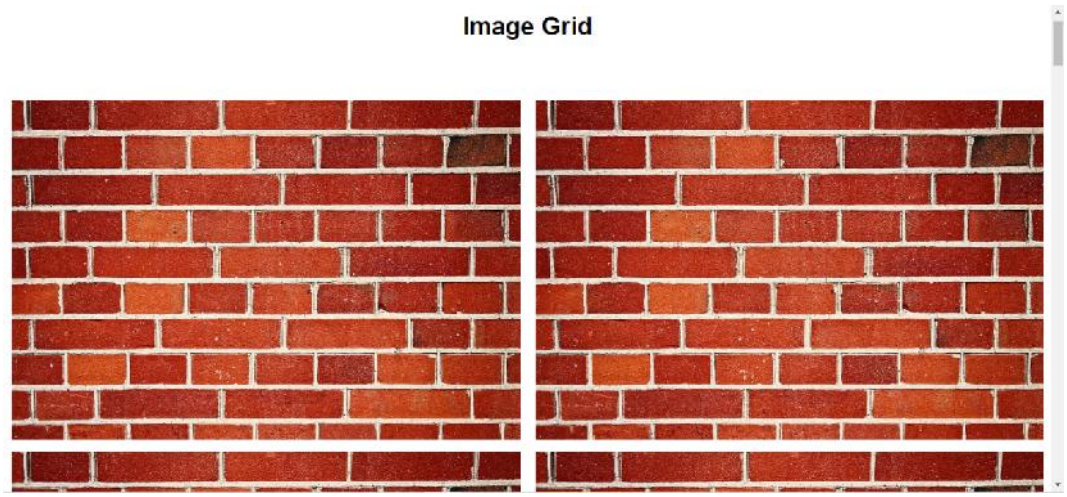


Figure 6

Resizing the browser window will show the result as in Figure 7.

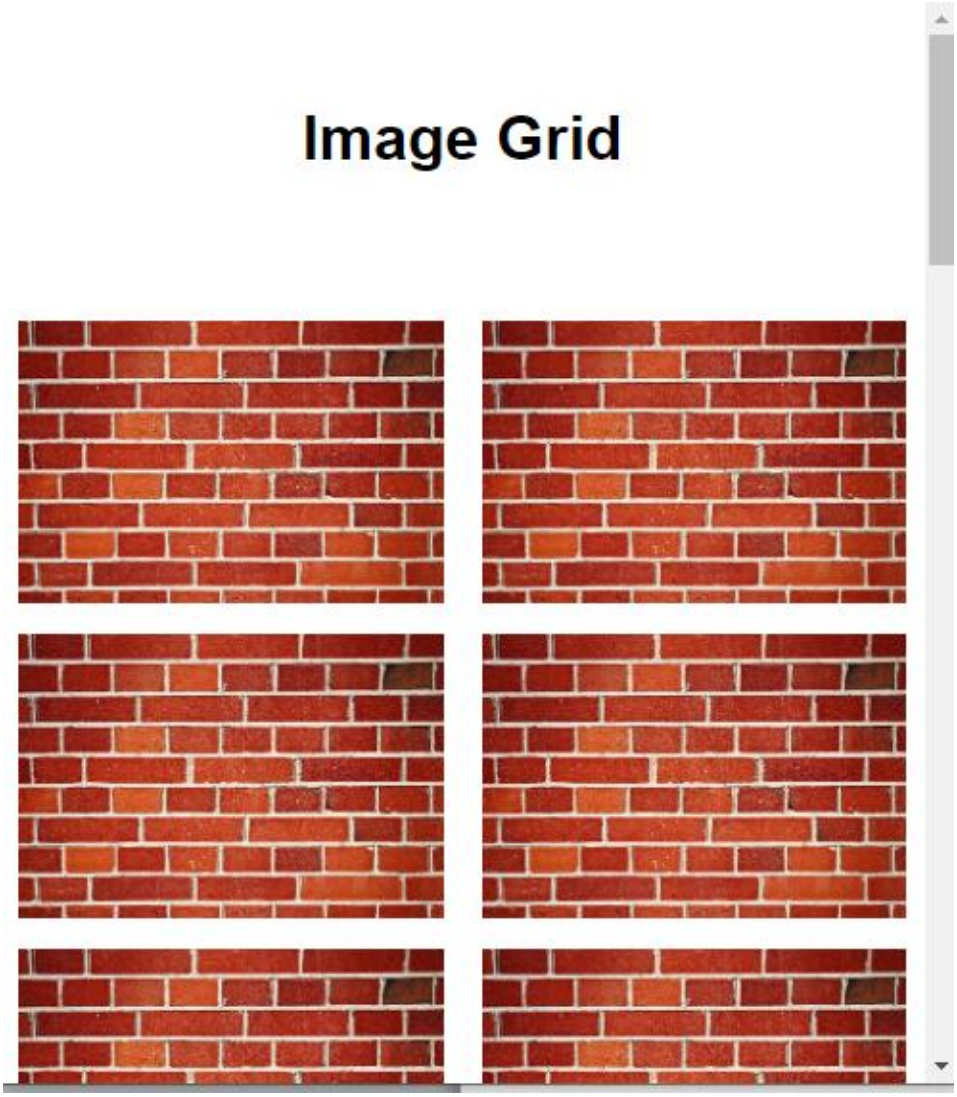


Figure 7

Summary

1. The <header> element represents a container for introductory content or a set of navigational links.
2. The <footer> HTML element represents a footer for its nearest ancestor sectioning content or sectioning root element

Keywords

- Header
- Footer
- Semantic tags

Self Assessment

1. Which one of the following contains information about the author?
   A. <footer>
   B. <header>
   C. <head>
   D. <body>

2. Header element does not contain _____
   A. logo
   B. <address>
   C. heading elements
   D. authorship information
3. Which element contains major navigational block?
   A. <nav>
   B. <address>
   C. <footer>
   D. <header>
4. Which element represents self-contained composition in document?
   A. <nav>
   B. <header>
   C. <footer>
   D. <article>
5. Which of the following element is used as a container for content?
   A. <aside>
   B. <article>
   C. <address>
   D. <footer>
6. Which element groups related content together?
   A. <aside>
   B. <footer>
   C. <section>
   D.
7. Which element works as a sidebar?
   A. header
   B. footer
   C. nav
   D. aside
8. Which HTML5 tag would you use to define footer?
   A. foot
   B. body
   C. bottom
   D. footer
9. Which of the following tag represents an independent piece of content of a document in HTML5?
   A. article
   B. section
   C. nav
   D. footer
10. Which of the following tags represents a section of a document used for navigation?
    A. navigation
    B. footer
    C. section
    D. nav
11. Which Semantic element is best suitable for content, like a blog post, that is self-contained, independent, and can be republished?
    A. div
    B. aside
    C. section
    D. article
12. The ____ element contains hyperlinks to other web pages within a website and is commonly positioned immediately after the closing </header> tag
    A. <head>...</head>
    B. <header>...</header>
    C. <nav>...</nav>
    D. <body>...</body>
13. Contains a specific grouping of content on the webpage.
    A. <head>...</head>
    B. <main>...</main>
    C. <section>...</section>

D. <header>...</header>

14. Contains the _____ content of the webpage.
    A. <footer>...</footer>
    B. <header>...</header>
    C. <main>...</main>
    D. <body>...</body>
15. Pick the odd one out
    A. <header>
    B. <nav>
    C. <footer>
    D. <title>

Answers for Self Assessment

l.	A	2.	B	3.	A	4.	A	5.	A
6.	C	7.	D	8.	D	9.	A	10.	D
11.	D	12.	C	13.	C	14.	C	15.	D

Review Questions

1. Illustrate the importance of semantic tags.
2. Classify different semantic tags in HTML
3. Discuss the concept of header element with example.
4. Create a web page that must comprise header, footer and main elements.
5. Create an image gallery comprising of two columns.
6. Demonstrate the use of article element with example.
7. Which element contains the body of the page? Illustrate with an example.
8. Design a page comprising a navbar and a footer comprising an image.
9. Differentiate between header and footer element with example.
10. Differentiate between article and aside element with an example

Further Readings

https://www.semrush.com/blog/semantic-html5-guide/

Unit 7:Creating responsive websites with media query and images

CONTENTS

Objectives

1. Understand different types of media
2. Understand different types of media outputs
3. Understand and implement media query structure

Introduction

When you wish to alter your website or application based on a device's overall kind (such as print vs. screen) or particular qualities and attributes (such as screen resolution or browser viewport width).Without making any changes to your markup, media queries help you alter how your web pages appear on a variety of devices, including smartphones, tablets, desktop computers, and more. An expression that matches the type and circumstances of a certain media feature, such as device width or screen resolution, is included in a media query together with a media type that represents the media type.A logical expression, media query can be resolved as either true or false. If the media type supplied in the media query matches the kind of device the document is being shown on and all media query expressions are met, the query's result will be true. When a media query returns true, the target device is styled according to the relevant style sheet or style rules.

7.1 Media Queries

Different style rules for various media types may be defined thanks to the @media rule, which was added to CSS2. One set of style guidelines might be used for television-like displays, mobile devices, printers, computer screens, and so on. Except for print media, these media formats sadly never received much support from gadgets.CSS3's media queries expanded on the concept of media types introduced in CSS2 by looking at a device's capabilities rather than its type. Many items may be checked with media queries, including:

1. dimensions of the viewport

2. dimensions of the gadget
3. Is the tablet or phone in portrait or landscape mode?
4. resolution

7.2 Media types

The different media types include the following:

Value	Description
All	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

7.3 Target different types of output

Although websites are frequently seen on screens, CSS may also be used to layout them for other outputs. Your web pages may need to appear one way on a computer screen and another when printed. This is made feasible by querying media kinds. The background color in this illustration is set to grey. However, the background color should be clear if the page is printed. User printer ink is preserved in this way.

```
body {
color: black;
 background-color: grey;
}
@media print {
 body {
   background-color: transparent;
 }
}
```

If you don't specify any media type for your CSS, it will automatically have a media type value of all. These two blocks of CSS are equivalent:

```
body {
color: black;
 background-color: white;
}
```

```
@media all {
  body {
color: black;
```

```
    background-color: white;
  }
}
```

Query conditions

You can add conditions to media types. These are called media queries. The CSS is applied only if the media type matches and the condition is also true. These conditions are called media features.

This is the syntax for media queries:

```
@media type and (feature)
```

Let's say you want to apply different styles depending on whether the browser window is in landscape mode (the viewport width is greater than its height) or portrait mode (the viewport height is greater than its width). There's a media feature called orientation you can use to test that:

```
@media all and (orientation: landscape) {
  // Styles for landscape mode.
}
@media all and (orientation: portrait) {
  // Styles for portrait mode.
}
```

In this case the media type is all. Because that's the default value, you can leave it out if you want:

```
@media (orientation: landscape) {
  // Styles for landscape mode.
}
@media (orientation: portrait) {
  // Styles for portrait mode.
}
```

Example:

```
<style>
@media screen and (orientation: landscape) {
 body::after {
  content: "Landscape";
 }
}
@media screen and (orientation: portrait) {
 body::after {
  content: "Portrait";
 }
```

```
}
body {
  margin: 0;
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
  width: 100vw;
  font-size: 10vmax;
}
</style>
```

The output generated by above example will be Figure 1 if the browser window is of full size (landscape)

Figure 1

The output changes to Figure 2 once the browser window is reduced to a portrait format,

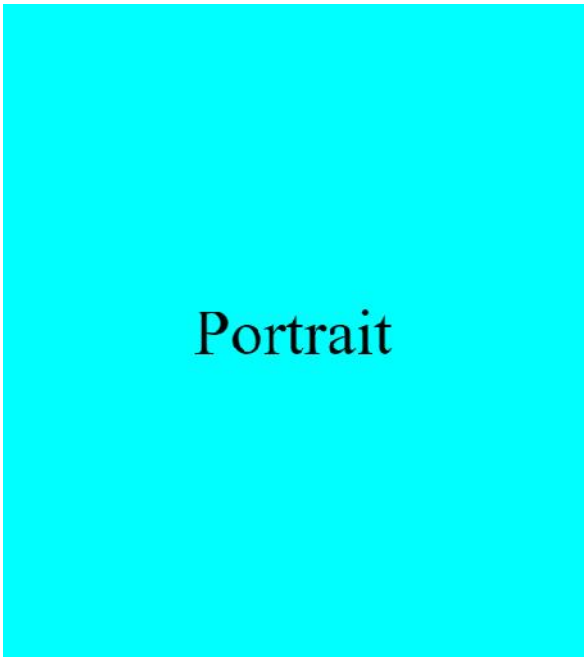


Figure 2

Adjust styles based on viewport size

For responsive design, one of the most useful media features involves the dimensions of the browser viewport. To apply styles when the browser window is wider than a certain width, use min-width.

```
@media (min-width: 400px) {
  // Styles for viewports wider than 400 pixels.
}
```

Use the max-width media feature to apply styles below a certain width:

```
@media (max-width: 400px) {
  // Styles for viewports narrower than 400 pixels.
}
```

You can also combine media queries to apply more than one condition. Use the word and to combine your media queries:

```
@media (min-width: 50em) and (max-width: 60em) {
  // Styles for viewports wider than 50em and narrower than 60em.
}
```

The examples are shown in Section 7.6.

7.4 Media Query Structure

A media type and one or more expressions that can resolve to true or false make up a media query.

```
@media not|onlymediatype and (expressions) {
  CSS-Code;
}
```

Refer to Figure 3 for a better understanding.

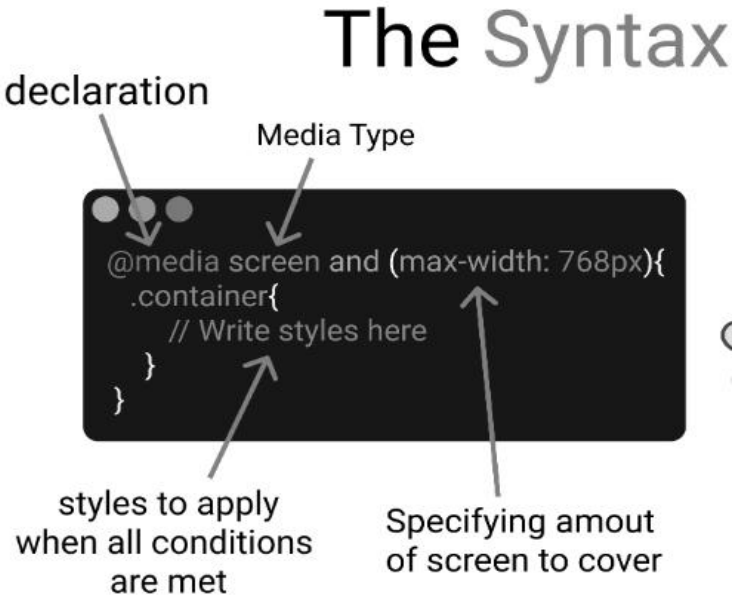


Figure 3

If the media type supplied matches the type of device the document is being shown on and all of the media query's expressions are true, the result of the query is true. A media query that returns true applies the associated style sheet or style rules in accordance with the standard cascade rules. The media type is optional, and the all type will be presumed unless the not or only operators are used. Additionally, you may use several stylesheets for various media:

```
<link rel="stylesheet" media="mediatypeand|not|only (expressions)" href="print.css">
```

7.5 Features of Media query

There are many features of media query which are listed below:

1. Color: The output device's bit count for each color component.
2. grid: Verifies whether the device is a bitmap or a grid.
3. height: The height of the viewport.
4. Aspect ratio: The proportion of the viewport's width to height.
5. color-index: The quantity of colors the gadget is capable of showing.
6. max-resolution: The device's dpi and dpcm maximum resolution.
7. monochromatic: A monochrome device's bit-per-color count.
8. Scanner: A device that scans outputs.
9. update: The output device's rate of modification.
10. width: The width of the viewport.

7.6 Media Queries examples

Including an alternative CSS section directly inside your style sheet is one approach to leverage media queries. The background colour in the example below is changed to lightgreen if the viewport is 600 pixels wide or more (pink if the viewport is less than 600 pixels wide):

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: pink;
}
@media screen and (min-width: 600px) {
  body {
    background-color: lightgreen;
  }
}
</style>
</head>
<body>

<h1>Resize the browser window to see the effect!</h1>
<p>The media query will only apply if the media type is screen and the viewport is 480px wide or wider.</p>

</body>
</html>
```

The output of above example is shown in Figure 4.

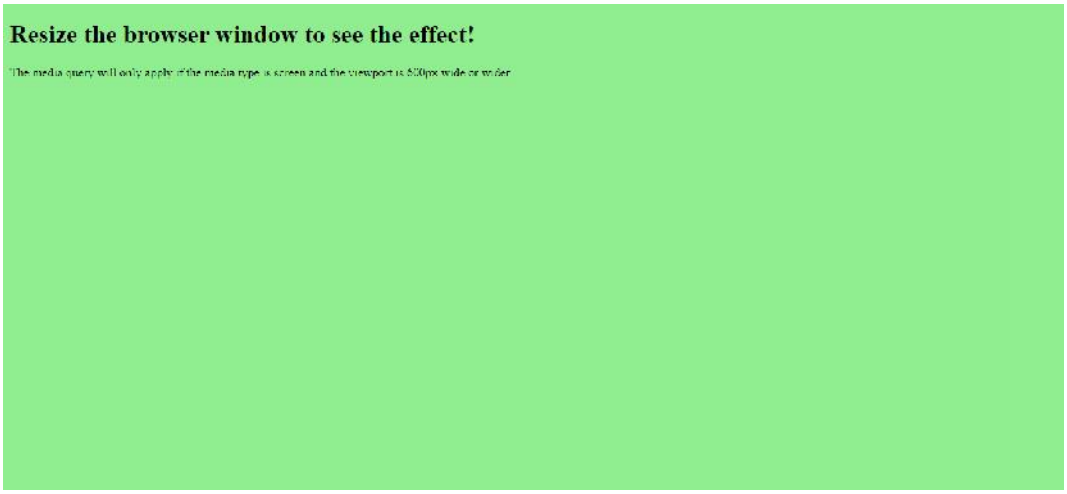


Figure 4

When the size of the browser window reduces below 600px, following is the output (as in Figure 5).

Figure 5

If the viewport is 600 pixels wide or wider, like in the following example, the menu will float to the left of the page; otherwise, it will be on top of the content.

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
.wrapper {overflow: auto;}
#main {margin-left: 4px;}
#leftsidebar {
  float: none;
  width: auto;
}
#menulist {
  margin: 0;
  padding: 0;
}
.menuitem {
  background: lightgreen;
  border: 1px solid #d4d4d4;
```

```
  border-radius: 4px;
  list-style-type: none;
  margin: 4px;
  padding: 2px;
height: 60px;
}
@media screen and (min-width: 600px) {
  #leftsidebar {width: 200px; float: left;}
  #main {margin-left: 216px;}
}
</style>
</head>
<body>
<div class="wrapper">
<div id="leftsidebar">
<ul id="menulist">
<li class="menuitem">Menu-item 1</li>
<li class="menuitem">Menu-item 2</li>
<li class="menuitem">Menu-item 3</li>
<li class="menuitem">Menu-item 4</li>
<li class="menuitem">Menu-item 5</li>
</ul>
</div>

<div id="main">
<h1>Resize the browser window to see the effect!</h1>
<p>This example shows a menu that will float to the left of the page if the viewport is 600 pixels wide or wider. If the viewport is less than 600 pixels, the menu will be on top of the content.</p>
</div>
</div>
</body>
</html>
```

The output of above example is shown in Figure 6.

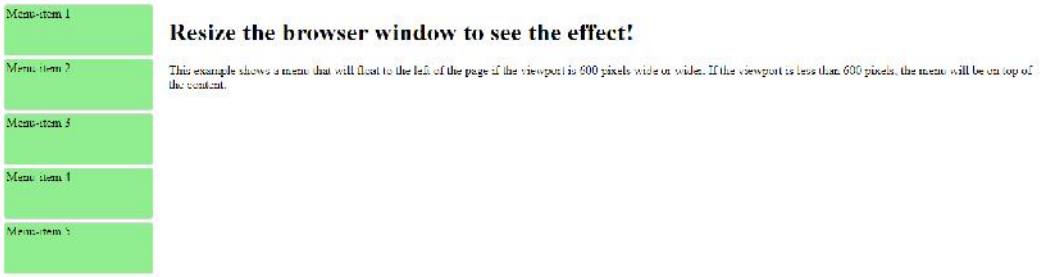


Figure 6

When the browser window falls below 600px, the following output (Figure 7) is generated.

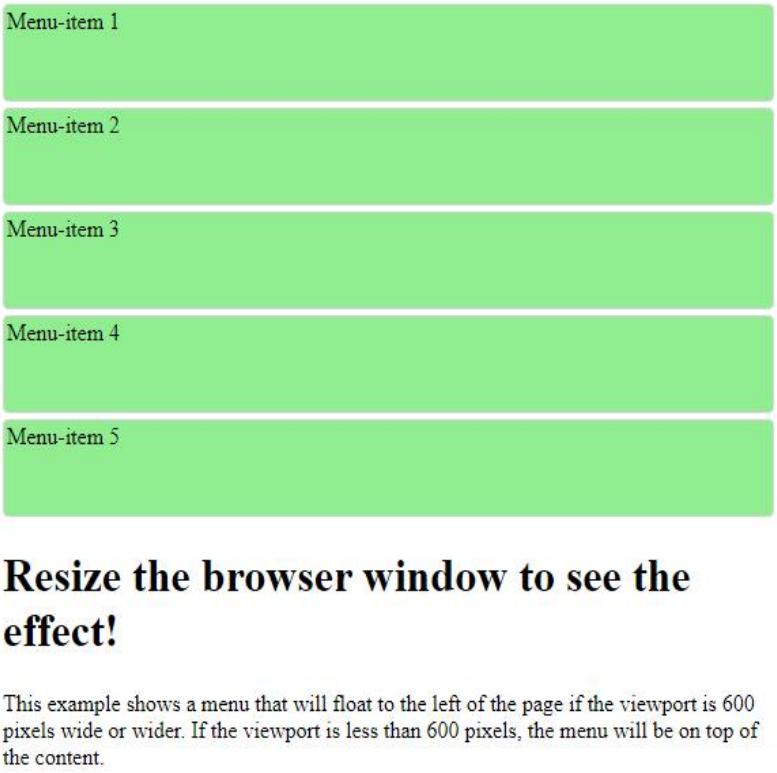


Figure 7

Summary

1. Different media types include all, print, screen and speech
2. Output using media queries can be achieved on the basis of media type, query conditions and viewport size

Keywords

- Media query
- Media

Self Assessment

1. A media query consists of a media type and zero or more expressions that check for the conditions of particular media features. State true or false.
   A. True
   B. False
2. Which of the following @viewport Property locks the document in the specified orientation, portrait or landscape?
   A. orientation
   B. resolution
   C. landscape
   D. portrait
3. Media type for computer screens, tablets, smart-phones is ___.
   A. print
   B. screen
   C. monitor
   D. window
4. If we want the background color to be yellow when the page is printed, which of the following is correct?

A. @media all {
   i. body {
   ii. background-color: yellow;
   iii. }
   b. }

B. @media screen {
   i. body {
   ii. background-color: yellow;
   iii. }
   b. }

C. @media print {
   i. body {
   ii. background-color: yellow;
   iii. }
   b. }

D. None

5. If you don't specify any media type for your CSS, it will automatically have a media type value of ___.
   A. screen
   B. print
   C. speech
   D. all
6. The viewport width is greater than its height. Which kind of orientation is this?
   A. portrait
   B. landscape
7. To apply styles when the browser window is wider than a certain width, use____.
   A. min-width
   B. max-width
8. To apply styles when the browser window is below a certain width, use____.
   A. min-width
   B. max-width
9. Which of the following is a feature of media query
   A. Color
   B. Grid
   C. Height
   D. All of the above
10. What does the following code do?

```
@media screen and (min-width: 700px) {
body {
background-color: red;
 }
}
```

A. The background color is changed to red if the viewport is 700 pixels wide or more.
B. The background color is changed to red if the viewport is 700 pixels wide or less.
C. Will do nothing
D. Compilation error

11. What does the following code do?

```
@media screen and (max-width: 700px) {
body {
background-color: red;
 }
}
```

A. The background color is changed to red if the viewport is 700 pixels wide or more.
B. The background color is changed to red if the viewport is 700 pixels wide or less.
C. Will do nothing
D. Compilation error

12. Which media type is used for screen readers?
A. voice
B. print
C. text
D. speech
13. The syntax for media query is___.
A. @media type and (feature)
B. @media feature and (type)
C. @media(feature)
D. Both a and c
14. In @media rule, to specify the media type and feature at the same time, we use ___.
A. hyphen
B. space
C. and
D. semi colon
15. Can we adjust styles on the basis of viewport size?
A. Yes
B. No

Answers for Self Assessment

l.	A	2.	A	3.	B	4.	C	5.	D
6.	B	7.	A	8.	B	9.	D	10.	A
11.	B	12.	D	13.	D	14.	C	15.	A

Review Questions

1. Illustrate the purpose of media queries.
2. Classify different media types used in media queries.
3. Demonstrate the syntax of@media rule.
4. Demonstrate the concept of orientation in media queries.
5. Quote the features of media queries.
6. Illustrate the concept of @media all with an example.

7. Differentiate between @media screen and @media print with an example.
8. Differentiate between min-width and max-width with the help of an example.
9. Describe the purpose of min-width with the help of an example.
10. Describe the purpose of max-width with the help of an example.

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Unit 8: Creating responsive websites with media query and images

CONTENTS

Objectives

1. Understand different types of media
2. Understand different types of media outputs
3. Understand and implement media query structure

Introduction

In order to customise a website's layout and the visual appeal dependent on the user's device or browser, media queries are a crucial component of responsive web design. Developers may construct websites that offer the best user experience across a variety of devices, from small smartphones to giant desktop screens, by employing media queries.The theory behind media inquiries is simple. We can provide certain CSS rules that are activated when predetermined criteria are satisfied. The dimensions of the screen, its height, its orientation, and its resolution are frequently part of these requirements. Developers may produce fluid and adaptive designs by providing several styles and layout characteristics for various circumstances.

8.1 Media Breakpoints and Data Ranges

Breakpoints are the points at which a website's layout snaps to accommodate a changed viewport size. Breakpoints are defined so that the necessary UI components appear and disappear when the available space changes. They are used to reflow information for several devices, conditionally reveal UI components that would otherwise be hidden on smaller screens, and adapt to different rendering modes such as printer output.

Most websites use numerous important breakpoints to transition between core layouts. These are frequently connected with the mobile, tablet, and desktop device families. Each popular CSS framework has its own set of predefined breakpoints. The breakpoints you choose are determined by the exact design needs and the devices you wish to target. Breakpoints are frequently defined based on popular device types such as smartphones, tablets, and desktop displays. However, the breakpoints can be tailored to meet the individual requirements of a project.

Framework	Small (Mobile)	Medium (Tablet)	Large (Small Monitor)	Extra Large (Large Monitor)
Bootstrap	576px	768px	992px	1200px

Semantic UI	N/A	723px	933px	1127px
Tailwind	640px	768px	1024px	1280px

At around the same moment, each of the three frameworks transitions to the next layout. The one notable exception to this rule is Semantic UI, which, in contrast to Bootstrap and Tailwind, implemented bigger layouts sooner (at narrower viewport sizes). Below their smallest breakpoint, all the frameworks size elements to fill the entire width of the screen.

Here are some common media breakpoints used in responsive web design:

1. Small devices (e.g., smartphones):
   - Breakpoint: 0 to 576 pixels
   - Styles: Typically, a single column layout, larger fonts, and simplified navigation.

2. Medium devices (e.g., tablets):
   - Breakpoint: 577 to 992 pixels
   - Styles: Adjustments to the layout, such as a two-column grid or wider content containers.

3. Large devices (e.g., desktops and laptops):
   - Breakpoint: 993 to 1200 pixels
   - Styles: More complex layouts, potentially utilizing a multi-column grid system and additional design elements.
4. Extra-large devices (e.g., large desktop screens):
   - Breakpoint: 1201 pixels and above
   - Styles: Further refinements to accommodate larger screens, such as wider Contentcontainers or additional sidebar sections.

The genuine breakpoints should be chosen based on the particular design objectives and target market, thus keep in mind that these breakpoints are merely examples. Another common approach is to set breakpoints for certain sections or elements of a website rather than the entire layout.

```
/* Styles for small devices (up to 576 pixels) */
@media (max-width: 576px) {
  /* Styles for small screens, such as smartphones */
```

```
  .container {
    width: 100%;
    padding: 10px;
  }
  .menu {
    display: none;
  }
}
```

This section defines the styles for small devices with a maximum width of 576 pixels. The .container element will have a width of 100% (i.e., it will occupy the full width of the parent container) and a padding of 10 pixels. The .menu element will be hidden (display: none).

```
/* Styles for medium devices (577 to 992 pixels) */
@media (min-width: 577px) and (max-width: 992px) {
  /* Styles for medium-sized screens, such as tablets */
  .container {
    width: 80%;
    padding: 20px;
  }
  .menu {
    display: block;
  }
}
```

This section sets the styles for medium devices with a width between 577 and 992 pixels. The .container element will have a width of 80% and a padding of 20 pixels. The .menu element will be displayed as a block (display: block).

```
/* Styles for large devices (993 to 1200 pixels) */
@media (min-width: 993px) and (max-width: 1200px) {
  /* Styles for large screens, such as desktops */
  .container {
    width: 70%;
    padding: 30px;
  }
  .menu {
    display: block;
  }
}
```

This section defines the styles for large devices with a width between 993 and 1200 pixels. The .container element will have a width of 70% and a padding of 30 pixels. The .menu element will be displayed as a block.

```
/* Styles for extra-large devices (above 1200 pixels) */
@media (min-width: 1201px) {
  /* Styles for extra-large screens, such as large desktops */
```

```
  .container {
    width: 60%;
    padding: 40px;
  }
  .menu {
    display: block;
  }
}
```

This section specifies the styles for extra-large devices with a width above 1200 pixels. The .container element will have a width of 60% and a padding of 40 pixels. The .menu element will be displayed as a block.

```
<!DOCTYPE html>
<html>
<head>
<style>
  /* Styles for small devices (up to 576 pixels) */
  @media (max-width: 576px) {
    /* Styles for small screens, such as smartphones */
    .container {
      width: 100%;
      padding: 10px;
      background-color: lightblue;
    }
    .menu {
      display: none;
    }
  }

  /* Styles for medium devices (577 to 992 pixels) */
  @media (min-width: 577px) and (max-width: 992px) {
    /* Styles for medium-sized screens, such as tablets */
    .container {
      width: 80%;
      padding: 20px;
      background-color: lightgreen;
    }
    .menu {
      display: block;
    }
  }
```

```
  /* Styles for large devices (993 to 1200 pixels) */
  @media (min-width: 993px) and (max-width: 1200px) {
    /* Styles for large screens, such as desktops */
    .container {
      width: 70%;
      padding: 30px;
      background-color: lightyellow;
    }
    .menu {
      display: block;
    }
  }

  /* Styles for extra-large devices (above 1200 pixels) */
  @media (min-width: 1201px) {
    /* Styles for extra-large screens, such as large desktops */
    .container {
      width: 60%;
      padding: 40px;
      background-color: lightgray;
    }
    .menu {
      display: block;
    }
  }
</style>
</head>
<body>
<div class="container">
<h1>Responsive Design Example</h1>
<p>This is an example of responsive design using media breakpoints.</p>
<ul class="menu">
<li>Menu Item 1</li>
<li>Menu Item 2</li>
<li>Menu Item 3</li>
</ul>
</div>
</body>
</html>
```

In the above code, we have defined media queries with different styles for specific screen widths. The .container element represents the main content container, and the .menu element represents a navigation menu.

When you run this code and resize the browser window, you will notice the following changes:

- On small screens (up to 576 pixels), the .container element will take up the full width of the screen with a light blue background. The .menu element will be hidden.
- On medium-sized screens (577 to 992 pixels), the .container element will have a width of 80% and a light green background. The .menu element will be displayed as a block.
- On large screens (993 to 1200 pixels), the .container element will have a width of 70% and a light yellow background. The .menu element will be displayed as a block.
- On extra-large screens (above 1200 pixels), the .container element will have a width of 60% and a light gray background. The .menu element will be displayed as a block.

By utilizing media breakpoints in this way, you can create a responsive design that adapts to various screen sizes and provides an optimized user experience across different devices.

The output generated by above example HTML file with Media breakpoint will be in

Figure 1

Figure 2

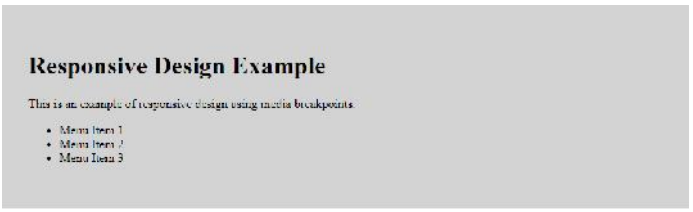


Figure 3

8.2 Media query handling Images

Using media queries, you can give different image sources or sizes based on the screen dimensions of the device. This will make the pictures work better on that device. Using CSS media queries, you can set breakpoints where the source or size of a picture changes. For smaller screens, you can load smaller images to reduce file size and speed up loading. For larger screens, you can load bigger or

higher-resolution images to make sure the pictures are clear and sharp. This method makes sure that the picture shown is optimized for each device. This improves performance, speeds up page loading, and gives the user a better overall experience.

```
<imgsrc="small-image.jpg" alt="Small Image" class="responsive-image">


<style>
.responsive-image {
  width: 100%;
  height: auto;
}


@media (min-width: 576px) {
  .responsive-image {
    /* Load larger image for screens with width 576 pixels or more */
    content: url(medium-image.jpg);
  }
}


@media (min-width: 992px) {
  .responsive-image {
    /* Load even larger image for screens with width 992 pixels or more */
    content: url(large-image.jpg);
  }
}
</style>
```

In the CSS code, the.responsive-image class is defined with a width of 100% and a height of auto, ensuring that the image resizes proportionally.

Inside the media queries, different images are loaded based on the screen width. For screens with a minimum width of 576px, the content property is used to set the image source to "medium-image.jpg". Similarly, for screens with a minimum width of 992px, the content property is used to set the image source to "large-image.jpg".

By using media queries, the appropriate image will be loaded based on the screen size, providing a responsive image experience. Make sure to adjust the image filenames and paths according to your actual image files.

Please note that the content property is typically used for pseudo-elements and generated content, and it may not work as intended for changing the src attribute of an <img> tag. To dynamically change the image source, you may need to use JavaScript.

```
<!DOCTYPE html>
<html>
<head>
<title>Image Function Example</title>
<style>
   /* Default styles */
img {
     display: block;
     width: 100%;
     height: auto;
```

```
  }

  /* Media queries using image functions */
  @media (width <= 480px) {
img {
      content: url("small-image.jpg");
    }
  }

  @media (width <= 768px) {
img {
      content: url("medium-image.jpg");
    }
  }

  @media (width <= 1024px) {
img {
      content: url("large-image.jpg");
    }
  }

  @media (width > 1024px) {
img {
      content: url("xlarge-image.jpg");
    }
  }
</style>
</head>
<body>
<imgsrc="default-image.jpg" alt="Default Image">
</body>
</html>
```

In the case above, the img element has a basic style with a width of 100% and a height of "auto" that makes it adaptable.

The content value is used by the different @media searches to set the picture URL based on the width of the screen. If the width of the screen is less than or equal to 480px, "small-image.jpg" will be loaded. If the width is less than or equal to 768px, "medium-image.jpg" will be loaded. If the width is less than or equal to 1024px, "large-image.jpg" will be loaded. Lastly, if the width is bigger than 1024px, "xlarge-image.jpg" will be loaded.

8.3 Media query Optimizing Images and Responsive Images

There are a few best practises to follow when optimising pictures for the web:

Resize and compress: Make sure your pictures are the right size for the web and have been optimised for it. Use picture editing software or web tools to resize them to the size you want and compress them to reduce file size without losing much quality.

Pick the right type of file: Choose the right file type for your pictures. JPEG is best for photos and images with a lot of colours and complicated shapes, while PNG is better for drawings, logos, and images with transparency. SVG (Scalable Vector images) could be used for simple icons or images with few colours.

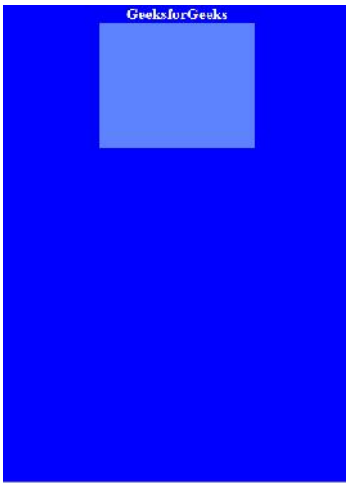
Use tools for image compression: Use tools and methods for image compression to shrink files even more. Tools like Kraken.io, TinyPNG, and ImageOptim can automatically reduce the size of pictures without making them look bad.

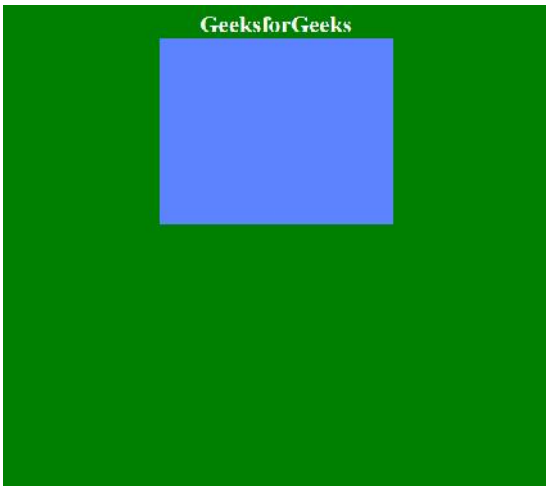
Lazy loading: If you want pictures on your website to load slowly, use lazy loading. Lazy loading waits to load pictures that are offscreen until the user scrolls to them. This makes the page load faster at first.

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Image Function Example</title>
<style>
   /* Default styles */
img {
     display: block;
     width: 100%;
     height: auto;
   }

   /* Media queries using image functions */
   @media (width <= 480px) {
img {
       content: url("small-image.jpg");
     }
   }

   @media (width <= 768px) {
img {
       content: url("medium-image.jpg");
     }
   }

   @media (width <= 1024px) {
img {
       content: url("large-image.jpg");
     }
   }

   @media (width > 1024px) {
img {
       content: url("xlarge-image.jpg");
     }
   }
</style>
</head>
<body>
<imgsrc="medium-image.jpg" alt="Default Image">
</body>
```

```
</html>
```





Summary

1. Different media types include all, print, screen and speech
2. Output using media queries can be achieved on the basis of media type, query conditions and viewport size

Keywords

- Media query
- Media
- Responsive Images
- Handling Images

Self Assessment

1. What is a media query in CSS used for?

A. Styling HTML elements
B. Adjusting layout based on screen sizes or device capabilities
C. Optimizing images for the web
D. Defining animations and transitions

2. Which attribute is used to specify multiple image sources with different resolutions in HTML?

A. srcset
B. media
C. sizes
D. alt

3. What is the purpose of lazy loading images on a webpage?

A. Reduce the initial page load time
B. Optimize images for different screen sizes
C. Apply different styles based on device capabilities
D. Enhance search engine optimization (SEO)

4. Which file format is typically recommended for photographs and complex images?

A. JPEG
B. PNG
C. GIF
D. SVG

5. How can you optimize images for the web?

A. Resize and compress images
B. Use responsive image techniques
C. Enable image caching
D. All of the above

6. Which CSS property allows an image to scale proportionally within its container?

A. width
B. height
C. object-fit

D. background-image

7. Which HTML element is used to provide different image sources based on screen size or device capabilities?

A. <img>
B. <picture>
C.
D. <source>

8. What is the purpose of defining different image dimensions in media queries?

A. To apply different styles to images
B. To optimize images for different screen sizes
C. To control the file size of images
D. To adjust the alignment of images

9. What is the primary purpose of using media queries in responsive web design?

a) To add animations to the website

b) To detect the user's device type

c) To create custom fonts for different screen sizes

d) To apply different styles based on the user's device or screen size

10. Which CSS property is commonly used to apply responsive styles with media queries?

a) font-size

b) display

c) background-color

d) @media

Answers for Self Assessment

l.	B	2.	A	3.	a	4.	A	5.	D
6.	C	7.	B	8.	B	9.	D	10.	D

Review Questions

1. What is the main purpose of media queries in CSS?
2. How can you optimize images for the web?
3. Which file format is commonly used for complex images and photographs?
4. What does lazy loading refer to in the context of images?
5. How can media queries be used to create responsive designs?

6. What HTML element is used to provide multiple image sources based on different conditions?
7. What are some image optimization techniques other than resizing and compressing?
8. How can you control the dimensions of an image using media queries?
9. What is the role of the srcset attribute in responsive images?
10. What are the benefits of using responsive images on a website?\

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Unit 9:Adding media queries to fluid layout

Objectives

1. Understand different types of media
2. Understand different types of media outputs
3. Understand and implement media query structure

Introduction

A fluid layout, also called a flexible layout, is a way of designing that lets a website or app's content change and adapt to fit different screen sizes and devices. Fluid layouts, as opposed to set layouts with fixed measurements, use relative units and flexible grids to make sure that parts adjust and move themselves in an equal way, giving users the same experience on all devices. By using media queries and CSS methods, fluid layouts can adapt to a wide range of screen sizes, from desktop computers to smartphones. This gives users a design that is easier to use and looks better, as well as one that fits their needs and preferences.

9.1 What is Fluid Layout in Media Query?

A flexible layout is a way of designing that lets content adapt and change size based on the amount of space on the screen. It uses relative units and flexible grids to ensure an adaptable design across different devices. Fluid styles make sure that the user experience stays the same, even when the screen size or position changes.

There are two main kinds of plans that are fluid:

Flexible Fluid Layout: In a flexible fluid layout, the lengths of the layout's parts change based on how much room is available. This is done by setting lengths with relative numbers, such as percentages. As the size of the window or container changes, the parts also change size. This type of fluid style has a design that stays the same and can be changed to fit different screen sizes.

Adaptive Fluid Layout: An adaptive fluid layout, also called a flexible layout, does more than just resize parts. It includes using CSS media queries to apply different styles based on certain breakpoints or sets of screen sizes. With customizable fluid layouts, you can define different sets of rules and change not only the width but also the general layout, placement, and style of parts to make a more personalised experience for different devices. This makes it easier to decide how the information is shown on different-sized screens.

It's important to note that these two types of flexible layouts don't contradict each other and can be used together to make a more complete and strong adaptable design. By using a mix of relative

units, fluid grids, and media queries, you can make a style that works well on a wide range of devices and screen resolutions and is very flexible and easy to use.

Here's an example of how you can add media queries to a fluid layout:

```
/* Default styles for all screen sizes */
.container {
width: 100%;
max-width: 960px;
margin: 0 auto;
padding: 20px;
}

/* Styles for screens larger than 768px */
@media (min-width: 768px) {
  .container {
padding: 40px;
  }
}

/* Styles for screens larger than 1200px */
@media (min-width: 1200px) {
  .container {
max-width: 1140px;
  }
}
```

In this example, the .container class represents the main container element of your fluid layout. By default, it has a width of 100%, a maximum width of 960 pixels, and some padding. This will ensure that the layout adjusts fluidly to different screen sizes.

The first media query is defined with @media (min-width: 768px) and targets screens larger than 768 pixels. Within this media query block, you can override the default styles and apply different styling, such as increasing the padding.

The second media query is defined with @media (min-width: 1200px) and targets screens larger than 1200 pixels. Here, you can further modify the .container styles, in this case, changing the maximum width.

You can add more media queries with different breakpoints to fine-tune the layout for various screen sizes. It's important to consider the needs of your specific design and adjust the styles accordingly.

Remember to include the appropriate meta tag in the <head> section of your HTML file to ensure proper viewport scaling:

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

This tag tells the browser to set the viewport width to the device width and scale the content accordingly, ensuring that your media queries work as intended on mobile devices.

9.2 Responsive Navigation Bar Design with Media Queries

It's important to create a menu bar that works well on different platforms by making it adaptable. With so many people using smartphones, tablets, and other devices with different screen sizes, it's important to make sure the menu bar fits the space without losing usefulness or looks.

Media queries are a key part of flexibility because they let us use different styles depending on the width of the screen. By using media queries, we can change the style of the menu bar on the fly, make it fill the width of the device, centre and shrink the name, and adjust the header text position.

A flexible menu bar that is well-made makes a website or app easier to use and more accessible. It also makes it easier for users to find their way around. It makes sure that important menu features are easy to find and are shown in the best way for each screen size, which improves the overall user experience.

In this chapter, we'll look at real-world cases and explain in detail how to use media queries to make flexible menu bars. By learning the ideas and methods behind flexible design, you will be able to make menu bars that fit smoothly on different devices. This will give you a user experience that works well and looks good.

```
<!DOCTYPE html>
<html>
<head>
<style>
   .navigation-bar {
background-color: #333;
color: #fff;
padding: 10px;
   }
      @media (max-width: 600px) {
    .navigation-bar {
width: 100%;
    }
   }
</style>
</head>
<body>
<div class="navigation-bar">
<!-- Navigation bar content goes here -->
</div>
</body>
</html>
```

It's important to create a menu bar that works well on different platforms by making it adaptable. With so many people using smartphones, tablets, and other devices with different screen sizes, it's important to make sure the menu bar fits the space without losing usefulness or looks.

Media queries are a key part of flexibility because they let us use different styles depending on the width of the screen. By using media queries, we can change the style of the menu bar on the fly, make it fill the width of the device, centre and shrink the name, and adjust the header text position.

A flexible menu bar that is well-made makes a website or app easier to use and more accessible. It also makes it easier for users to find their way around. It makes sure that important menu features are easy to find and are shown in the best way for each screen size, which improves the overall user experience.

In this chapter, we'll look at real-world cases and explain in detail how to use media queries to make flexible menu bars. By learning the ideas and methods behind flexible design, you will be able to make menu bars that fit smoothly on different devices. This will give you a user experience that works well and looks good.

9.3 Responsive centre and shrink logo in media query

In responsive web design, it is crucial to ensure that elements, such as logos, adapt appropriately to different screen sizes. When building a responsive navigation bar, it is common to center and shrink the logo to maintain a balanced and visually pleasing layout on smaller devices.

To achieve this effect, media queries come into play. Media queries allow us to define specific styles for different screen widths. By using a media query with a maximum width of 600 pixels (or any desired value), we can target smaller screens and apply specific styles to center and shrink the logo within the navigation bar.

The CSS code within the media query includes the following changes:

The .navigation-bar class has a width: 100%; property, making it expand horizontally to fill the entire width of the device.

The .logo class has a max-width: 80px; property, limiting the logo's width to a maximum of 80 pixels. This ensures that the logo maintains an appropriate size on smaller screens.

Additionally, the .logo class has a margin: auto; property, which automatically sets equal margins on the left and right sides. This centers the logo horizontally within the navigation bar.

These modifications result in the logo being centered and shrunk to a maximum width of 80 pixels when the screen size is 600 pixels or less. The combination of width: 100%; on the navigation bar and max-width: 80px; with margin: auto; on the logo allows for a visually balanced and responsive navigation bar design on smaller devices.

By utilizing media queries effectively, you can ensure that your navigation bar and its elements, such as logos, adapt seamlessly to different screen sizes, providing an optimal user experience across various devices.

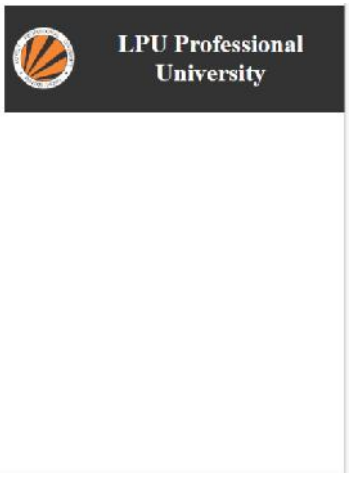
```
<!DOCTYPE html>
<html>
<head>
```

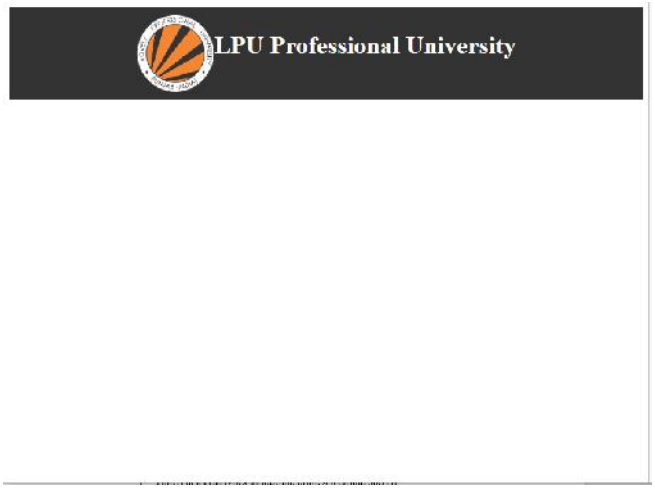
```
<style>
   .navigation-bar {
background-color: #333;
color: #fff;
padding: 10px;
display: flex;
justify-content: center;
   }

   .logo {
max-width: 100px;
height: auto;
   }

   .header {
text-align: center;
   }

   @media (max-width: 600px) {
     .navigation-bar {
width: 100%;
     }

     .logo {
max-width: 80px;
margin: auto;
     }
   }
</style>
</head>
<body>
<div class="navigation-bar">
<imgsrc="lpu1.png" alt="Logo" class="logo">
<h1 class="header">LPU Professional University</h1>
</div>
</body>
</html>
```

Within the media query, the .logo class has been updated with the margin: auto; property. This centers the logo horizontally within the navigation bar by setting automatic margins on both the left and right sides.

By incorporating this change, the logo will be centered and shrunk to a maximum width of 80 pixels when the screen size is 600 pixels or less. The max-width: 80px; property within the media query allows the logo to adjust and maintain an appropriate size for smaller screens.

Feel free to customize the max-width values and add more styles as needed to achieve your desired centering and shrinking effect for the logo.





When designing a responsive navigation bar, it is important to ensure that the text alignment of the header adapts appropriately to different screen sizes. By utilizing media queries, we can target specific screen widths and adjust the text alignment of the header to provide an optimal user experience.

In the provided code snippet, a media query with a maximum width of 600 pixels has been implemented. Within this media query, the text alignment for the header is adjusted using the .header class and the text-align property.

By default, outside the media query, the header text is centered using text-align: center;, ensuring it is horizontally centered within the navigation bar.

Within the media query, the .header class is updated with text-align: right;. This change aligns the header text to the right within the navigation bar when the screen size is 600 pixels or less. This adjustment allows for better utilization of the available space on smaller devices and ensures a visually pleasing layout.

You can customize the text alignment styles within the media query to suit your specific design requirements. By leveraging media queries to adjust the text alignment of the header, you can create a responsive navigation bar that offers a consistent and user-friendly experience across a range of devices and screen sizes.

```
<!DOCTYPE html>
<html>
<head>
<style>
  .navigation-bar {
background-color: #784;
color: #fff;
padding: 10px;
display: flex;
justify-content: center;
  }

  .logo {
max-width: 100px;
height: auto;
  }

  .header {
text-align: center;
  }

  @media (max-width: 600px) {
    .navigation-bar {
width: 100%;
    }

    .logo {
```

```
max-width: 80px;
margin: auto;
    }

    .header {
text-align: right;
    }
  }
</style>
</head>
<body>
<div class="navigation-bar">
<imgsrc="lpuonline.png" alt="Logo" class="logo">
<h1 class="header">LPU Online </h1>
</div>
</body>
</html>
```

Within the media query, the .header class has been updated with the text-align: right; property. This aligns the header text to the right within the navigation bar on screens with a width of 600 pixels or less.

By incorporating this change, the header text will be aligned to the right when the screen size is 600 pixels or less, allowing for better utilization of the available space on smaller devices.

Feel free to customize the text alignment styles and add more styles as needed to achieve your desired effect for the header text within the media query.

Summary

- Media queries play a crucial role in responsive web design by allowing the adjustment of styles based on screen size.
- The targeted screen width in this example is 600 pixels, which is a common threshold for smaller devices.
- The navigation bar is made to fill the width of the device by setting its width to 100% within the media query.

Keywords

- Media query
- Responsive web design
- Screen width
- Navigation bar
- Width adjustment

Self Assessment

1. What is the purpose of using a media query in responsive web design?
    A. To create a navigation bar
    B. To adjust layout and styles based on screen size
    C. To optimize image loading
    D. To improve search engine optimization

2. What is the targeted screen width in the media query for this example?
    A. 320px
    B. 600px
    C. 1024px
    D. 1440px

3. How is the navigation bar width adjusted in the media query?
    A. It is set to a fixed width of 600px.
    B. It is set to a percentage value based on the screen width.
    C. It remains the same as outside the media query.

D. It expands to fill the entire width of the device.

4. How is the logo centered within the navigation bar in the media query?
   A. By using the text-align: center; property.
   B. By setting margin: auto; on the logo.
   C. By using the float: center; property.
   D. By applying a flexbox layout to the navigation bar.

5. What is the purpose of shrinking the logo within the media query?
   A. To reduce image file size for faster loading.
   B. To fit the logo within the limited space on smaller screens.
   C. To improve the visibility of other navigation elements.
   D. To match the logo size with the header text.

6. How is the text alignment of the header adjusted within the media query?
   A. It is aligned to the left.
   B. It is aligned to the right.
   C. It remains centered.
   D. It is aligned to both the left and right edges.

7. Which CSS property is commonly used to adjust styles within a media query?
   A. color
   B. font-size
   C. display
   D. max-width

8. Why is it important to make the navigation bar fill the width of the device in a media query?
   A. To create a visually appealing design.
   B. To ensure the navigation bar is fully visible.
   C. To prevent horizontal scrolling on small screens.
   D. To optimize the website for search engine rankings.

9. How does a responsive navigation bar enhance user experience?
   A. It makes the website load faster.
   B. It improves search engine optimization.
   C. It adapts to different screen sizes and devices.
   D. It increases conversion rates.

10. How does the use of media queries contribute to responsive design?
    A. It enables dynamic layout adjustments based on screen size.
    B. It ensures compatibility with older web browsers.
    C. It speeds up the loading time of web pages.
    D. It prevents users from zooming in and out on mobile devices.

Answers for Self Assessment

1 B 2 B 3 D 4 B 5 B 6 A 7 D 8 C

9 C 10 A

Review Questions

1. What is the purpose of a media query in responsive web design?
2. What is the targeted screen width in the media query for this example?
3. How is the navigation bar width adjusted in the media query?
4. How is the logo centered within the navigation bar in the media query?
5. Why is it important to shrink the logo within the media query?
6. How is the text alignment of the header adjusted within the media query?
7. Which CSS property is commonly used to adjust styles within a media query?
8. Why is it important to make the navigation bar fill the width of the device in a media query?
9. How does a responsive navigation bar enhance user experience?
10. How does the use of media queries contribute to responsive design?

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Unit 10:Adding media queries to fluid layout

Objectives

1. Understand different types of media
2. Understand different types of media outputs
3. Understand and implement media query structure

Introduction

A fluid layout, also called a flexible layout, is a way of designing that lets a website or app's content change and adapt to fit different screen sizes and devices. Fluid layouts, as opposed to set layouts with fixed measurements, use relative units and flexible grids to make sure that parts adjust and move themselves in an equal way, giving users the same experience on all devices. By using media queries and CSS methods, fluid layouts can adapt to a wide range of screen sizes, from desktop computers to smartphones. This gives users a design that is easier to use and looks better, as well as one that fits their needs and preferences.

10.1What is Fluid Layout in Media Query?

A flexible layout is a way of designing that lets content adapt and change size based on the amount of space on the screen. It uses relative units and flexible grids to ensure an adaptable design across different devices. Fluid styles make sure that the user experience stays the same, even when the screen size or position changes.

There are two main kinds of plans that are fluid:

Flexible Fluid Layout: In a flexible fluid layout, the lengths of the layout's parts change based on how much room is available. This is done by setting lengths with relative numbers, such as percentages. As the size of the window or container changes, the parts also change size. This type of fluid style has a design that stays the same and can be changed to fit different screen sizes.

Adaptive Fluid Layout: An adaptive fluid layout, also called a flexible layout, does more than just resize parts. It includes using CSS media queries to apply different styles based on certain breakpoints or sets of screen sizes. With customizable fluid layouts, you can define different sets of rules and change not only the width but also the general layout, placement, and style of parts to make a more personalised experience for different devices. This makes it easier to decide how the information is shown on different-sized screens.

It's important to note that these two types of flexible layouts don't contradict each other and can be used together to make a more complete and strong adaptable design. By using a mix of relative units, fluid grids, and media queries, you can make a style that works well on a wide range of devices and screen resolutions and is very flexible and easy to use.

Here's an example of how you can add media queries to a fluid layout:

```
/* Default styles for all screen sizes */
.container {
  width: 100%;
  max-width: 960px;
  margin: 0 auto;
  padding: 20px;
}

/* Styles for screens larger than 768px */
@media (min-width: 768px) {
  .container {
    padding: 40px;
  }
}

/* Styles for screens larger than 1200px */
@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
  }
}
```

In this example, the .container class represents the main container element of your fluid layout. By default, it has a width of 100%, a maximum width of 960 pixels, and some padding. This will ensure that the layout adjusts fluidly to different screen sizes.

The first media query is defined with @media (min-width: 768px) and targets screens larger than 768 pixels. Within this media query block, you can override the default styles and apply different styling, such as increasing the padding.

The second media query is defined with @media (min-width: 1200px) and targets screens larger than 1200 pixels. Here, you can further modify the .container styles, in this case, changing the maximum width.

You can add more media queries with different breakpoints to fine-tune the layout for various screen sizes. It's important to consider the needs of your specific design and adjust the styles accordingly.

Remember to include the appropriate meta tag in the <head> section of your HTML file to ensure proper viewport scaling:

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

This tag tells the browser to set the viewport width to the device width and scale the content accordingly, ensuring that your media queries work as intended on mobile devices.

10.2Enhancing Responsive Design: Optimizing Footer

In the realm of responsive web design, optimizing the footer plays a crucial role in creating a seamless user experience across various screen sizes. The footer, located at the bottom of a web page, typically contains important information, links, and supplementary content. Enhancing the footer's responsiveness involves adjusting its content to fill the available width on different devices.

By applying responsive techniques, the footer content can be intelligently adjusted to adapt to various screen sizes. This may involve utilizing CSS properties like display: flex or grid to create a responsive grid system that evenly distributes the content and expands it to fill the available width. Additionally, media queries can be employed to make specific adjustments to the footer layout and styling based on different screen breakpoints.

Furthermore, optimizing the footer may include enhancing the visibility of its content on smaller devices. This can be achieved by employing techniques such as font size adjustments, increasing spacing between links, or utilizing collapsible sections to efficiently manage content without overwhelming the user interface.

The goal of enhancing the footer in responsive design is to ensure that it remains informative, visually appealing, and user-friendly across a range of devices. By implementing effective responsive design strategies, the footer can seamlessly adapt to different screen sizes, providing an optimized user experience and ensuring that users can access essential information regardless of the device they are using.

```
<!DOCTYPE html>
<html>
<head>
<style>
   .footer {
     background-color: #333;
     color: #fff;
     padding: 20px;
     text-align: center;
   }

   @media (max-width: 600px) {
     .footer {
       display: flex;
       flex-wrap: wrap;
       justify-content: center;
     }
   }
</style>
</head>
<body>
<div class="content">
<center><h1>Welcome to Lovely Professional University</h1></center>
<imgsrc="lpu.png" alt="LPU Logo" width="300" height="100">
```

```
<p align="justify">Lovely Professional University (LPU) is a private university situated in Punjab, India. It is the largest single-campus university in India, offering a wide range of undergraduate and postgraduate programs. LPU focuses on providing quality education and a conducive learning environment to its students.</p>

<p align="justify">At LPU, students can benefit from state-of-the-art facilities, experienced faculty, and a vibrant campus life. The university places a strong emphasis on industry exposure, entrepreneurship, and practical learning through internships and industry tie-ups.</p>

</div>


<footer class="footer">

<p>&copy; 2023 Lovely Professional University. All rights reserved.</p>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

<p>Located in Punjab, India</p>

</footer>

</body>

</html>
```

The element represents the website's main content area.

The <h1> heading welcomes users to Lovely Professional University (LPU).

The <img> tag includes the LPU logo (replace "lpu-logo.png" with the actual image file).

The <footer class="footer"> element represents the footer section of the webpage.

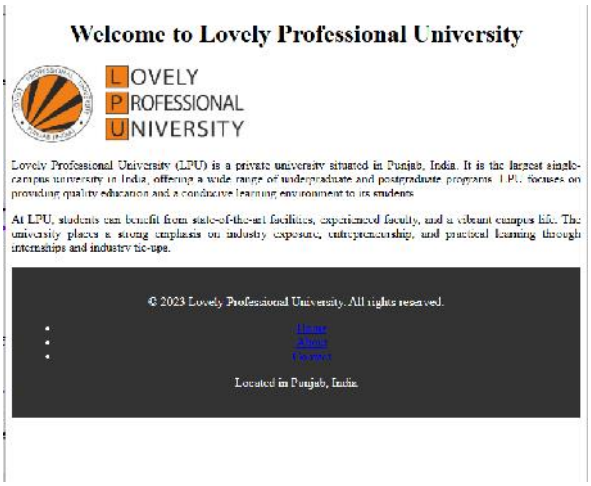
CSS styles are applied to the .footer class to provide a visually appealing and consistent design.

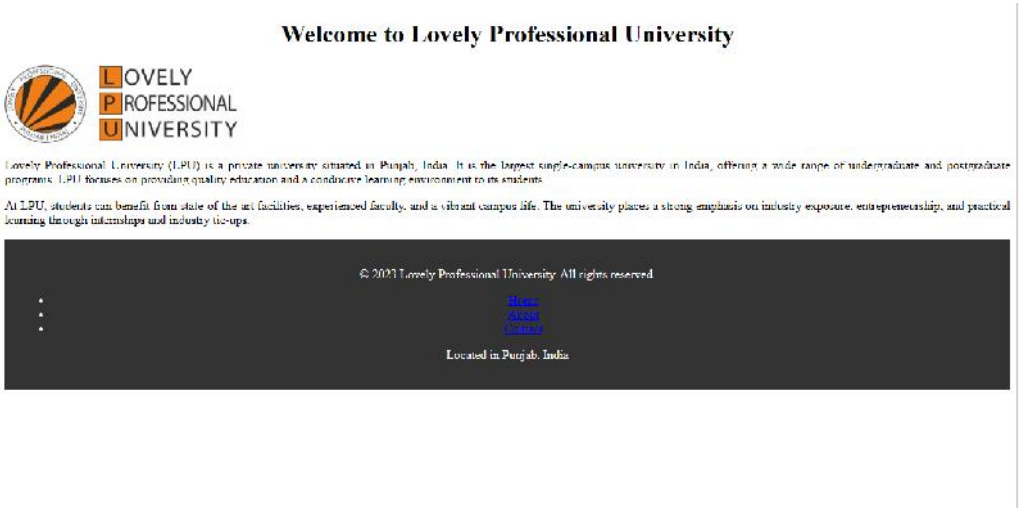
The footer includes a <p> tag with a copyright notice mentioning Lovely Professional University and the current year.

A <nav> element contains an unordered list (<ul>) with links for navigation, such as "Home," "About," and "Contact."

A final <p> tag provides additional information, in this case, stating the university's location in Punjab, India.

By using media queries, the footer's responsiveness is enhanced. In this example, the media query specifies a maximum screen width of 600 pixels. Inside the media query, the footer layout is adjusted using flexbox properties. The .footer class is modified to use display: flex; and justify-content: center;, ensuring the footer content is centered and responsive on smaller devices.





10.3Responsive paragraph content in the four columns

To provide a description of the paragraph content in the four columns:

HTML Structure:

The paragraph content is organized within four columns, typically represented by elements or any other suitable container. Each column may have its own class, such as , to facilitate styling and targeting.

Paragraph Content:

Within each column, there are paragraphs of text that provide relevant information, descriptions, or any desired content. The paragraphs may be written in HTML using <p> tags, allowing for proper formatting and readability.

Styling and Customization:

The paragraph content can be customized using CSS to match the desired design and layout of the webpage. We can apply various styles, such as font size, color, margin, or padding, to enhance the visual appearance of the paragraphs. Additionally, CSS frameworks or libraries like Bootstrap can be utilized to further streamline the styling process and create responsive column layouts.

By organizing the content into four columns with paragraph elements, you can present information in a structured and visually pleasing manner. This arrangement allows for better readability and organization, enabling users to easily comprehend and navigate through the provided information.

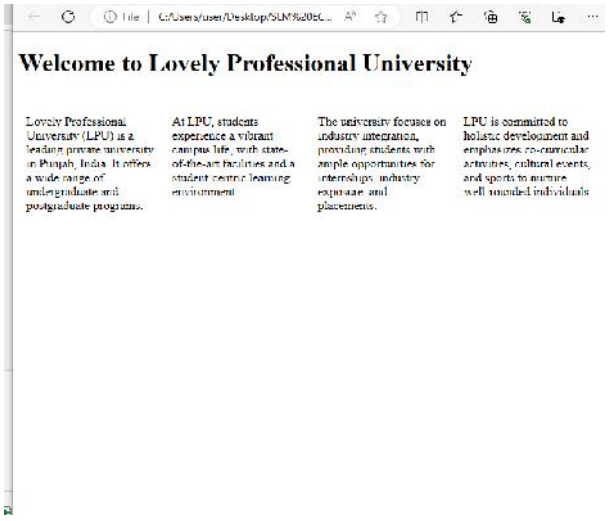
```
<!DOCTYPE html>
<html>
<head>
<style>
  .column {
    width: 25%;
    float: left;
    padding: 10px;
    box-sizing: border-box;
  }

  @media (max-width: 600px) {
    .column {
      width: 100%;
      float: none;
    }
  }
</style>
</head>
<body>
<div class="content">
<h1>Welcome to Lovely Professional University</h1>
<div class="column">
<p>Lovely Professional University (LPU) is a leading private university in Punjab, India. It offers a wide range of undergraduate and postgraduate programs.</p>
</div>
<div class="column">
<p>At LPU, students experience a vibrant campus life, with state-of-the-art facilities and a student-centric learning environment.</p>
```

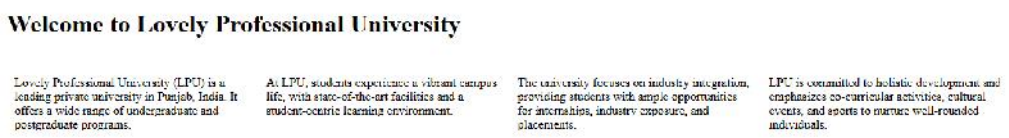
```
</div>
<div class="column">
<p>The university focuses on industry integration, providing students with ample opportunities for internships, industry exposure, and placements.</p>
</div>
<div class="column">
<p>LPU is committed to holistic development and emphasizes co-curricular activities, cultural events, and sports to nurture well-rounded individuals.</p>
</div>
</div>
</body>
</html>
```

In above example, the paragraph content is organized into four columns using elements with the class column. Each column contains a <p> tag for the respective paragraph content.

The CSS styles define the layout and properties for the columns. By default, the columns are set to a width of 25% and floated left, allowing them to align horizontally. The padding and box-sizing properties are used to manage spacing and ensure the box model behaves as expected.

Within the media query, specified with (max-width: 600px), the columns are adjusted to occupy the full width and the float property is set to none. This ensures that on screens with a width of 600 pixels or less, the columns stack vertically, creating a responsive layout suitable for smaller devices.





10.4 Responsive adjusting the navigation content forvisibility on a small device

In responsive web design, adjusting the navigation content for visibility on a small device is crucial to provide an optimal user experience. This involves modifying the navigation layout and behavior to ensure easy access to navigation links on smaller screens.To achieve this, a media query is used to target the specific screen width where the adjustment should occur. Typically, the media query sets a maximum width, such as 600 pixels, which is commonly associated with small devices like mobile phones.Within the media query, the navigation content is adjusted using CSS styles. One common approach is to hide the regular navigation links and replace them with a more compact and mobile-friendly menu icon or button.

In the example provided, the navigation content is initially designed as a flex container (display: flex;) with the logo positioned to the left and an unordered list (ul) representing the menu positioned to the right. The menu itself is initially set to display: none;, effectively hidden.However, within the media query, the menu is set to display: block;, making it visible on small devices. Meanwhile, the unordered list (ul) containing the navigation links is set to display: none;, effectively hiding it on small devices.

By implementing this approach, the navigation content is adjusted dynamically based on screen size. On small devices, the regular navigation links are hidden, and a more compact menu representation becomes visible. This ensures better visibility and usability of the navigation on smaller screens, enhancing the overall user experience.It's important to note that this is just one example of adjusting the navigation for small devices. The specific implementation may vary depending on design preferences and requirements.

```
<!DOCTYPE html>
<html>
<head>
<style>
  .navigation {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #333;
color: #fff;
    padding: 10px;
  }
```

```
  .menu {
    display: none;
  }

  @media (max-width: 600px) {
    .menu {
      display: block;
    }
    .navigation ul {
      display: none;
    }
  }
</style>
</head>
<body>
<div class="navigation">
<h1>Logo</h1>
<ul class="menu">
<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>
</div>
</body>
</html>
```

In above example, the navigation content is adjusted for visibility on small devices within a media query targeting a maximum width of 600 pixels. Initially, the navigation is designed as a flex container (display: flex;) with the logo positioned to the left and an unordered list (ul) representing the menu positioned to the right. Within the media query, the menu is set to display: block;, making it visible on small devices. Simultaneously, the unordered list (ul) containing the navigation links is set to display: none;, hiding it on small devices.

By implementing this approach, the navigation menu is replaced with a menu icon or another suitable UI element on small devices, ensuring better visibility and usability for users on mobile or smaller screens. Feel free to customize the styles, layout, and content of the navigation menu to match your specific design requirements.

Summary

- Adjust the footer content to fill the width, hide
- the paragraph content in the four columns, adjust the navigation content to make it
- more visible on a small deviceThe targeted screen width in this example is 600 pixels, which is a common threshold for smaller devices.
- The navigation bar is made to fill the width of the device by setting its width to 100% within the media query.

Keywords

Media query

Responsive web design

Paragraph content

Four columns

Adjust navigation

Visibility

Small device

Self Assessment

1. What is the objective of adjusting the footer content to fill the width?
   A. Improve website performance
   B. Enhance visual appeal
   C. Ensure responsiveness
   D. Optimize search engine ranking

2. How can you ensure the footer content fills the entire width of the device?
   A. Apply a fixed width to the footer
   B. Use media queries to set the width to 100%
   C. Increase the padding of the footer
   D. Decrease the margin of the footer

3. Why would you hide the paragraph content in the four columns?
   A. To save space on the webpage
   B. To improve website security
   C. To enhance search engine optimization
   D. To reduce page load time

4. Which CSS property can be used to hide the paragraph content?
   A. display: visible;
   B. visibility: hidden;
   C. display: none;
   D. opacity: 0;

5. What is the purpose of adjusting the navigation content to make it more visible on a small device?
   A. Improve website accessibility
   B. Enhance user engagement
   C. Ensure consistency across devices
   D. Optimize website speed

6. How can you make the navigation content more visible on a small device?
   A. Increase the font size of the navigation links
   B. Use media queries to change the navigation layout
   C. Add a background color to the navigation bar
   D. Implement a scrollable navigation menu

7. Which property is commonly used to replace regular navigation links on a small device?
   A. display: flex;
   B. display: block;
   C. display: inline;

D. display: none;

8. What is the benefit of using media queries for adjusting the navigation content?
   A. Improves website security
   B. Enhances search engine optimization
   C. Allows for targeted styling based on screen size
   D. Reduces website maintenance efforts

9. Which element is often used to represent a more compact and mobile-friendly navigation menu on small devices?
   A.
   B. <a>
   C. <button>
   D. <h1>

10. How does adjusting the navigation content for small devices contribute to a better user experience?
    A. Improves website aesthetics
    B. Enhances website credibility
    C. Facilitates easier navigation on mobile devices
    D. Increases website traffic

Answers for Self Assessment

1 C 2 B 3 A 4 C 5 A 6 B 7 D 8 C

9 C 10 C

Review Questions

1 What is the purpose of adjusting the footer content to fill the width?

2 How can you hide the paragraph content in the four columns?

3 What is the objective of adjusting the navigation content for better visibility on a small device?

4 How can you make the navigation content more visible on a small device?

5 Which CSS property can be used to hide an element from the webpage?

6 What is the benefit of using media queries for adjusting the navigation content?

7 What is a common element used to represent a more compact navigation menu on small devices?

8 Why is it important to ensure responsiveness in web design?

9 How can media queries help in creating a responsive layout?

10 What are the advantages of optimizing the navigation for small devices?

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Unit 11:Working Responsively

Objectives

Understand the importance of prioritizing content before layout in mobile design to deliver effective and engaging user experiences.

Comprehend the principles and techniques of responsive design and its significance in ensuring optimal viewing experiences across various devices and screen sizes.

Identify and utilize appropriate design and prototyping tools for creating mobile interfaces, enabling efficient and streamlined development processes.

Recognize the key elements of user experience (UX) in the context of mobile and beyond, focusing on navigation optimization, accessibility, and iterative design for enhanced user satisfaction.

Introduction

In today's world, more and more people are accessing the web on mobile devices. This means that it's more important than ever to create websites that are responsive to different screen sizes and orientations. Responsive design is a web design approach that uses fluid grids and flexible layouts to ensure that a website looks good and functions well on any device. This means that you don't need to create separate mobile and desktop versions of your website.

There are a number of tools that can help you create responsive websites. Some of the most popular include:

Bootstrap: Bootstrap is a popular front-end framework that includes a number of responsive CSS and HTML components.

Foundation: Foundation is another popular front-end framework that offers a wide range of responsive features.

Semantic UI: Semantic UI is a framework that uses semantic markup to create accessible and understandable code.

In addition to using the right tools, it's also important to consider the user experience when designing responsive websites. This means making sure that the content is easy to read and the navigation is intuitive, regardless of the device being used.

11.1Creating content before and after use of responsive design tools

Before the advent of responsive design tools, creating content for the web was a bit of a challenge. If you wanted your website to look good on both desktop and mobile devices, you had to create two separate versions of your content, one for each device. This was a time-consuming and error-prone

process. To create content for desktop devices, you would use a fixed-width layout. This meant that the width of your content would be the same on every device, regardless of the screen size. This could lead to problems on mobile devices, where the content would be too wide to fit on the screen.

To create content for mobile devices, you would use a fluid-width layout. This meant that the width of your content would adjust to the screen size of the device. This was a better solution than a fixed-width layout, but it still required a lot of manual work to ensure that the content looked good on all devices.

Creating content after responsive design tools

The introduction of responsive design tools made it much easier to create content that looked good on all devices. These tools use media queries to dynamically adjust the layout of your content based on the screen size of the device. This means that you only need to create one version of your content, and it will automatically look good on all devices.

There are a number of responsive design tools available, including Bootstrap, Foundation, and Semantic UI. These tools provide a number of features that make it easy to create responsive content, including:

Fluid grids: Fluid grids allow you to create layouts that adjust to the screen size of the device.

Flexible images: Flexible images resize to fit the width of the container.

Media queries: Media queries allow you to specify different layouts for different screen sizes.

Using responsive design tools makes it easy to create content that looks good on all devices. This is a major advantage, as it means that you don't have to create separate versions of your content for each device. This can save you a lot of time and effort, and it can also improve the user experience for your visitors.

Here are some tips for creating content that looks good on all devices:

Use clear and concise language. This will make your content easier to read on small screens.

Use large fonts. This will also make your content easier to read on small screens.

Use images and videos sparingly. Large images and videos can slow down loading times on mobile devices.

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>My Responsive Website</title>
</head>
<body>
<h1>This is my responsive website</h1>
<p>This text will look good on all devices, regardless of the screen size.</p>
<imgsrc="image.jpg" alt="Image">
</body>
</html>
```

In the above example code uses a <meta> tag to set the viewport to the width of the device. This means that the website will automatically adjust its layout to fit the screen size of the device.

The <p> tag also uses a media query to specify a different font size for mobile devices. This means that the text will be larger on mobile devices, making it easier to read.

The <img> tag uses a srcset attribute to specify different images for different screen sizes. This means that the website will automatically load the appropriate image for the device being used.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>My Responsive Website</title>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
</head>
<body>
<div class="container">
<h1>This is my responsive website</h1>
<p class="lead">This text will look good on all devices, regardless of the screen size.</p>
<imgsrc="image.jpg" alt="Image">
</div>
</body>
</html>
```

This code uses Bootstrap to create a responsive website. Bootstrap is a popular front-end framework that includes a number of responsive CSS and HTML components.

The <meta> tag sets the viewport to the width of the device. This means that the website will automatically adjust its layout to fit the screen size of the device.

The tag creates a container that will be used to hold the content of the website. This container will be responsive, meaning that it will adjust its width to fit the screen size of the device.

The <h1> and <p> tags use Bootstrap classes to style the text. These classes will ensure that the text looks good on all devices, regardless of the screen size.

The <img> tag uses a srcset attribute to specify different images for different screen sizes. This means that the website will automatically load the appropriate image for the device being used.

11.2Responsive Design and User Experience

Responsive design in HTML involves creating a layout that adapts and responds to different screen sizes, ensuring optimal user experience across devices. This is achieved through the use of CSS media queries, which allow you to define specific styles for different screen widths or device capabilities. User experience (UX) focuses on designing interfaces that are intuitive, accessible, and visually appealing, providing a seamless interaction for users.

```
<!DOCTYPE html>
<html>
<head>
<title>Lovely Professional University - Responsive Design Example</title>
```

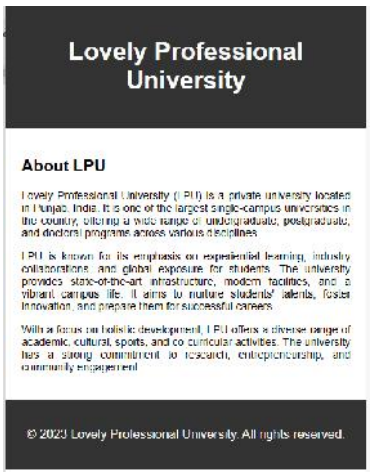
```
<style>
  /* Default styles for the website */
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
  }

  .header {
    background-color: #333;
color: #fff;
    padding: 20px;
    text-align: center;
  }

  .content {
    padding: 20px;
    text-align: justify;
  }

  .footer {
    background-color: #333;
color: #fff;
    padding: 20px;
    text-align: center;
  }

  /* Media query for small screens */
  @media screen and (max-width: 600px) {
    .header, .footer {
      font-size: 16px;
    }

    .content {
      font-size: 14px;
    }
  }
</style>
</head>
<body>
```

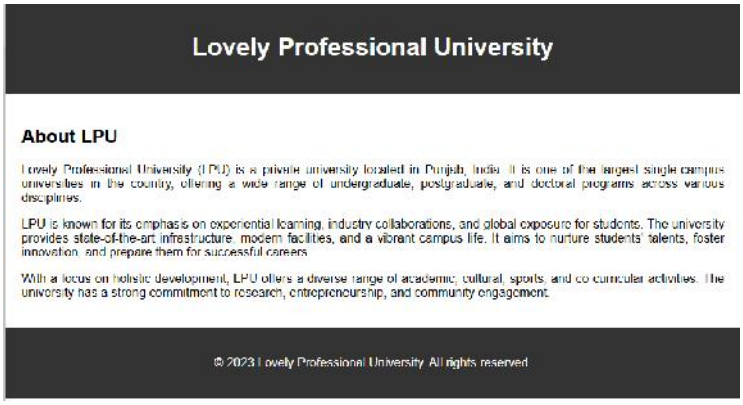
```
<div class="header">
<h1>Lovely Professional University</h1>
</div>

<div class="content">
<h2>About LPU</h2>
<p>Lovely Professional University (LPU) is a private university located in Punjab, India. It is one of the largest single-campus universities in the country, offering a wide range of undergraduate, postgraduate, and doctoral programs across various disciplines.</p>
<p>LPU is known for its emphasis on experiential learning, industry collaborations, and global exposure for students. The university provides state-of-the-art infrastructure, modern facilities, and a vibrant campus life. It aims to nurture students' talents, foster innovation, and prepare them for successful careers.</p>
<p>With a focus on holistic development, LPU offers a diverse range of academic, cultural, sports, and co-curricular activities. The university has a strong commitment to research, entrepreneurship, and community engagement. </p>
</div>

<div class="footer">
<p>&copy; 2023 Lovely Professional University. All rights reserved. </p>
</div>
</body>
</html>
```

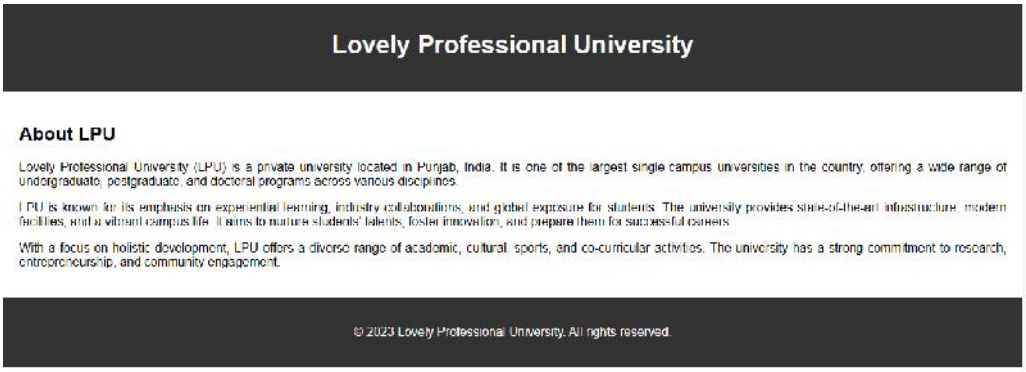
In the above example, we have a simple webpage layout consisting of a header, content section, and footer. The CSS styles define the default appearance of the elements. The media query is used to adjust the styles for small screens (max-width: 600px), providing a responsive design.

In this case, the font sizes are adjusted for smaller screens to improve readability. The header and footer elements have a larger font size (16px), while the content section has a smaller font size (14px). This ensures that the text remains legible and the overall layout adapts gracefully to smaller screens.

By using media queries and adjusting styles based on screen size, we achieve a responsive design that enhances the user experience. The webpage adapts its layout and typography to different devices, providing an optimal viewing experience and maintaining readability.







11.3Responsive Design layout in different devices

In responsive web design, adjusting the navigation content for visibility on a small device is crucial to provide an optimal user experience. This involves modifying the navigation layout and behavior to ensure easy access to navigation links on smaller screens.To achieve this, a media query is used to target the specific screen width where the adjustment should occur. Typically, the media query sets a maximum width, such as 600 pixels, which is commonly associated with small devices like mobile phones.Within the media query, the navigation content is adjusted using CSS styles. One common approach is to hide the regular navigation links and replace them with a more compact and mobile-friendly menu icon or button.

In the example provided, the navigation content is initially designed as a flex container (display: flex;) with the logo positioned to the left and an unordered list (ul) representing the menu positioned to the right. The menu itself is initially set to display: none;, effectively hidden.However, within the media query, the menu is set to display: block;, making it visible on small devices. Meanwhile, the unordered list (ul) containing the navigation links is set to display: none;, effectively hiding it on small devices.

By implementing this approach, the navigation content is adjusted dynamically based on screen size. On small devices, the regular navigation links are hidden, and a more compact menu representation becomes visible. This ensures better visibility and usability of the navigation on smaller screens, enhancing the overall user experience.It's important to note that this is just one example of adjusting the navigation for small devices. The specific implementation may vary depending on design preferences and requirements.

```
<!DOCTYPE html>
<html>
<head>
<title>Computer Company - Responsive Design Layout Example</title>
<style>
```

```
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
  }

  .container {
    max-width: 960px;
    margin: 0 auto;
    padding: 20px;
  }

  .header {
    background-color: #333;
color: #fff;
    padding: 20px;
    text-align: center;
  }

  .content {
    padding: 20px;
    text-align: justify;
  }

  .footer {
    background-color: #333;
color: #fff;
    padding: 20px;
    text-align: center;
  }

  @media screen and (max-width: 768px) {
    .container {
      padding: 10px;
    }

    .header, .footer {
      font-size: 16px;
    }
  }
```

```
  @media screen and (max-width: 480px) {
    .container {
      padding: 5px;
    }

    .header, .footer {
      font-size: 14px;
    }
  }
</style>
</head>
<body>
<div class="container">
<div class="header">
<h1>Computer Company</h1>
</div>

<div class="content">
<h2>Welcome to Computer Company!</h2>
<p>Computer Company is a leading provider of cutting-edge computer hardware and software solutions. We offer a wide range of products, including high-performance desktop computers, laptops, peripherals, and software applications.</p>
<p>Our team of experts is dedicated to delivering top-quality products that meet the needs of professionals, businesses, and enthusiasts alike. We strive to provide innovative solutions that enhance productivity, efficiency, and creativity.</p>
<p>Whether you're a gaming enthusiast, a business professional, or a student, Computer Company has the perfect technology solutions to empower your work and leisure. Experience the latest advancements in computer technology with us.</p>
</div>

<div class="footer">
<p>&copy; 2023 Computer Company. All rights reserved.</p>
</div>
</div>
</body>
</html>
```

In this example, the responsive design layout is customized for a computer-related company. The HTML structure includes a header, content, and footer sections, contained within a element with the class "container" to provide a maximum width of 960 pixels and center the layout on the page.

The CSS styles define the appearance of the elements, including background colors, padding, and text alignment. Media queries are used to adjust the layout for different screen sizes, with font sizes decreasing for smaller devices to enhance readability.

You can customize the content within the section to highlight the unique offerings, services, or products of your computer-related company. Additionally, feel free to modify the styling and add relevant images or interactive elements to create a visually appealing and engaging user experience.

Summary

- Prioritize content over layout by understanding user needs and goals. Deliver concise, well-structured content that engages users effectively.
- Implement responsive design techniques to ensure optimal viewing experiences across different devices and screen sizes. Use CSS media queries to adapt the layout and elements based on breakpoints.
- Utilize design and prototyping tools like Sketch, Figma, or Adobe XD to create mobile interfaces efficiently. Development frameworks such as React Native or Flutter help build responsive and cross-platform mobile applications.
- Focus on delivering a seamless user experience. Optimize navigation for mobile screens, ensure fast loading times, implement touch-friendly interactions, personalize content, and prioritize accessibility.
- Iterate and test designs with real users to refine the mobile experience further. Incorporate user feedback and usability testing to identify areas for improvement and enhance the overall user experience.

Keywords

- Content-first approach
- Responsive design
- CSS media queries
- Design and prototyping tools
- Development frameworks
- User experience (UX)

Self Assessment

1. Which approach emphasizes organizing content effectively for mobile experiences?
    A. Layout before content
    B. Content before layout
    C. Design before content
    D. Content alongside layout

2. What is the purpose of responsive design in mobile development?
    A. Optimizing content for desktop devices only
    B. Adapting layouts to different screen sizes and devices
    C. Creating fixed-width designs for specific devices
    D. Ignoring the need for mobile optimization

3. Which CSS feature allows you to define different styles based on screen sizes?
    A. Flexbox
    B. Media queries
    C. Grid layout
    D. Float positioning

4. Which of the following is a popular design and prototyping tool for mobile interfaces?
    A. Photoshop
    B. Illustrator
    C. Sketch
    D. InDesign

5. Which development frameworks enable building cross-platform mobile applications?
    A. AngularJS
    B. React Native
    C. jQuery Mobile
    D. Django

6. What does user experience (UX) focus on in mobile and beyond?
    A. Optimizing loading times
    B. Enhancing visual aesthetics
    C. Delivering a seamless interaction
    D. Increasing website traffic

7. What is an essential consideration for mobile navigation optimization?
    A. Increasing the number of navigation links
    B. Using complex gestures for interaction
    C. Minimizing the number of clicks required
    D. Displaying navigation menus in a sidebar

8. Why is accessibility important in mobile design?
    A. It helps increase website rankings in search engines.
    B. It ensures compatibility with various web browsers.
    C. It improves user engagement and satisfaction.
    D. It allows for faster loading times.

9. What does iterative design involve in mobile development?
    A. Creating a static design without user feedback
    B. Implementing user feedback to refine the design
    C. Skipping the testing phase for quicker development
    D. Making design decisions based on personal preferences

10. What is the purpose of usability testing in mobile design?

A. Identifying areas for improvement in the user experience
B. Enhancing the visual appeal of the mobile interface
C. Increasing the speed and performance of the mobile app
D. Providing a detailed analysis of coding techniques

Answers for Self Assessment

1 B 2 B 3 B 4 C 5 B 6 C 7 C 8 C

9 B 10 A

Review Questions

1. What is the main principle behind content before layout in mobile design?
2. What is responsive design and why is it important for mobile experiences?
3. How do media queries contribute to responsive design?
4. Name one popular design and prototyping tool used for mobile interfaces.
5. Which development frameworks facilitate building cross-platform mobile applications?
6. What does user experience (UX) encompass in the context of mobile and beyond?
7. Why is navigation optimization crucial for mobile interfaces?
8. What is the significance of accessibility in mobile design?
9. Explain the concept of iterative design in the context of mobile development.
10. What is the purpose of usability testing in mobile design?

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Unit 12:Working Responsively

Objectives

Understand the importance of prioritizing content before layout in mobile design to deliver effective and engaging user experiences.

Comprehend the principles and techniques of responsive design and its significance in ensuring optimal viewing experiences across various devices and screen sizes.

Identify and utilize appropriate design and prototyping tools for creating mobile interfaces, enabling efficient and streamlined development processes.

Recognize the key elements of user experience (UX) in the context of mobile and beyond, focusing on navigation optimization, accessibility, and iterative design for enhanced user satisfaction.

Introduction

In today's world, more and more people are accessing the web on mobile devices. This means that it's more important than ever to create websites that are designed for mobile devices. Traditionally, websites were designed for desktop computers. This meant that they were often too large and complex for mobile devices. However, with the rise of mobile devices, it's become increasingly important to create websites that are designed for mobile first. Mobile-first design is a web design approach that prioritizes the mobile experience. This means that the website is designed to be used on mobile devices first, and then adapted for desktop computers. There are a number of benefits to mobile-first design. First, it ensures that your website is accessible to a wider audience. Second, it can improve the user experience for mobile users. Third, it can save you time and money in the long run. In this chapter, we will discuss the importance of mobile-first design and how to create mobile-friendly websites. We will also discuss some of the tools and techniques that you can use to create mobile-first websites.

Why is mobile-first design important?

There are a number of reasons why mobile-first design is important. First, as mentioned above, more and more people are accessing the web on mobile devices. In fact, according to a recent study by Statista, over 50% of web traffic now comes from mobile devices.

Second, mobile users are different from desktop users. They tend to use their devices in different ways and have different expectations. For example, mobile users are more likely to use their devices one-handed, so it's important to make sure that your website is easy to use with one hand.

Third, mobile-first design can improve the user experience for mobile users. This is because mobile-first websites are designed to be used on mobile devices first, so they are more likely to be optimized for mobile devices.

12.1Creating content before and after use of responsive design tools

Before the advent of responsive design tools, creating content for the web was a bit of a challenge. If you wanted your website to look good on both desktop and mobile devices, you had to create two separate versions of your content, one for each device. This was a time-consuming and error-prone process. To create content for desktop devices, you would use a fixed-width layout. This meant that the width of your content would be the same on every device, regardless of the screen size. This could lead to problems on mobile devices, where the content would be too wide to fit on the screen.

To create content for mobile devices, you would use a fluid-width layout. This meant that the width of your content would adjust to the screen size of the device. This was a better solution than a fixed-width layout, but it still required a lot of manual work to ensure that the content looked good on all devices.

Creating content after responsive design tools

The introduction of responsive design tools made it much easier to create content that looked good on all devices. These tools use media queries to dynamically adjust the layout of your content based on the screen size of the device. This means that you only need to create one version of your content, and it will automatically look good on all devices.

There are a number of responsive design tools available, including Bootstrap, Foundation, and Semantic UI. These tools provide a number of features that make it easy to create responsive content, including:

Fluid grids: Fluid grids allow you to create layouts that adjust to the screen size of the device.

Flexible images: Flexible images resize to fit the width of the container.

Media queries: Media queries allow you to specify different layouts for different screen sizes.

Using responsive design tools makes it easy to create content that looks good on all devices. This is a major advantage, as it means that you don't have to create separate versions of your content for each device. This can save you a lot of time and effort, and it can also improve the user experience for your visitors.

Here are some tips for creating content that looks good on all devices:

Use clear and concise language. This will make your content easier to read on small screens.

Use large fonts. This will also make your content easier to read on small screens.

Use images and videos sparingly. Large images and videos can slow down loading times on mobile devices.

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>My Responsive Website</title>
</head>
<body>
<h1>This is my responsive website</h1>
<p>This text will look good on all devices, regardless of the screen size.</p>
<imgsrc="image.jpg" alt="Image">
</body>
</html>
```

In the above example code uses a <meta> tag to set the viewport to the width of the device. This means that the website will automatically adjust its layout to fit the screen size of the device.

The <p> tag also uses a media query to specify a different font size for mobile devices. This means that the text will be larger on mobile devices, making it easier to read.

The <img> tag uses a srcset attribute to specify different images for different screen sizes. This means that the website will automatically load the appropriate image for the device being used.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>My Responsive Website</title>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
</head>
<body>
<div class="container">
<h1>This is my responsive website</h1>
<p class="lead">This text will look good on all devices, regardless of the screen size.</p>
<imgsrc="image.jpg" alt="Image">
</div>
</body>
</html>
```

This code uses Bootstrap to create a responsive website. Bootstrap is a popular front-end framework that includes a number of responsive CSS and HTML components.

The <meta> tag sets the viewport to the width of the device. This means that the website will automatically adjust its layout to fit the screen size of the device.

The tag creates a container that will be used to hold the content of the website. This container will be responsive, meaning that it will adjust its width to fit the screen size of the device.

The <h1> and <p> tags use Bootstrap classes to style the text. These classes will ensure that the text looks good on all devices, regardless of the screen size.

The <img> tag uses a srcset attribute to specify different images for different screen sizes. This means that the website will automatically load the appropriate image for the device being used.

12.2Responsive Design and User Experience

Responsive design in HTML involves creating a layout that adapts and responds to different screen sizes, ensuring optimal user experience across devices. This is achieved through the use of CSS media queries, which allow you to define specific styles for different screen widths or device

capabilities. User experience (UX) focuses on designing interfaces that are intuitive, accessible, and visually appealing, providing a seamless interaction for users.

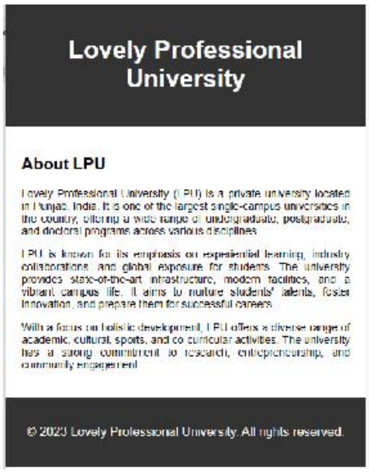
```
<!DOCTYPE html>
<html>
<head>
<title>Lovely Professional University - Responsive Design Example</title>
<style>
   /* Default styles for the website */
body {
font-family: Arial, sans-serif;
margin: 0;
padding: 0;
   }

   .header {
background-color: #333;
color: #fff;
padding: 20px;
text-align: center;
   }

   .content {
padding: 20px;
text-align: justify;
   }

   .footer {
background-color: #333;
color: #fff;
padding: 20px;
text-align: center;
   }

   /* Media query for small screens */
   @media screen and (max-width: 600px) {
     .header, .footer {
font-size: 16px;
     }

     .content {
```

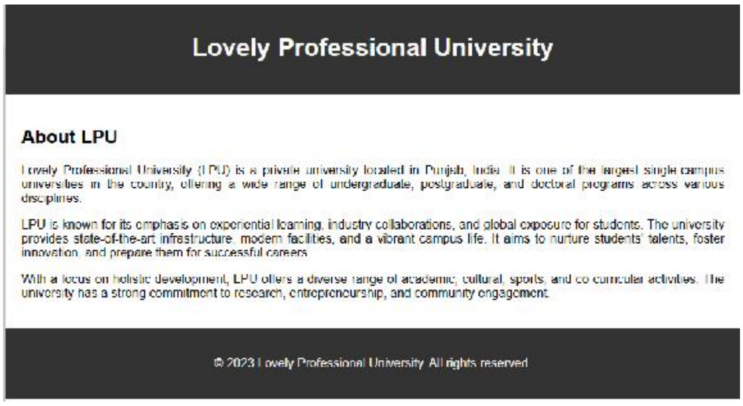
```
font-size: 14px;
    }
  }
</style>
</head>
<body>
<div class="header">
<h1>Lovely Professional University</h1>
</div>

<div class="content">
<h2>About LPU</h2>
<p>Lovely Professional University (LPU) is a private university located in Punjab, India. It is one of the largest single-campus universities in the country, offering a wide range of undergraduate, postgraduate, and doctoral programs across various disciplines.</p>
<p>LPU is known for its emphasis on experiential learning, industry collaborations, and global exposure for students. The university provides state-of-the-art infrastructure, modern facilities, and a vibrant campus life. It aims to nurture students' talents, foster innovation, and prepare them for successful careers.</p>
<p>With a focus on holistic development, LPU offers a diverse range of academic, cultural, sports, and co-curricular activities. The university has a strong commitment to research, entrepreneurship, and community engagement. </p>
</div>

<div class="footer">
<p>&copy; 2023 Lovely Professional University. All rights reserved. </p>
</div>
</body>
</html>
```

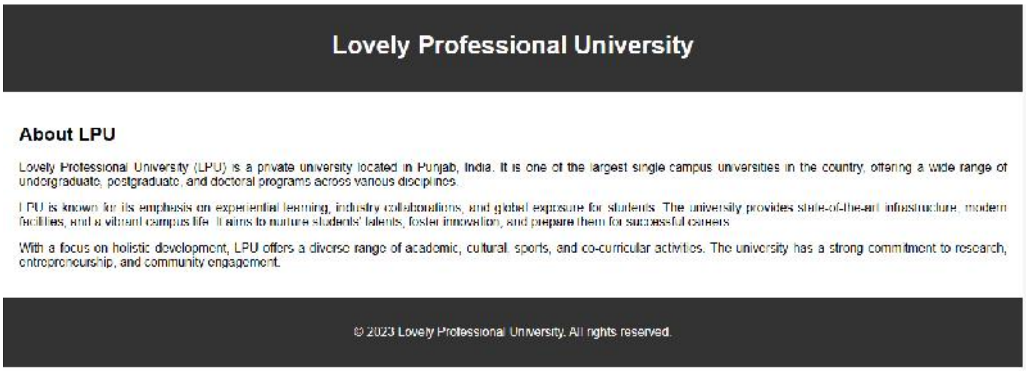
In the above example, we have a simple webpage layout consisting of a header, content section, and footer. The CSS styles define the default appearance of the elements. The media query is used to adjust the styles for small screens (max-width: 600px), providing a responsive design.

In this case, the font sizes are adjusted for smaller screens to improve readability. The header and footer elements have a larger font size (16px), while the content section has a smaller font size (14px). This ensures that the text remains legible and the overall layout adapts gracefully to smaller screens.

By using media queries and adjusting styles based on screen size, we achieve a responsive design that enhances the user experience. The webpage adapts its layout and typography to different devices, providing an optimal viewing experience and maintaining readability.







12.3 Responsive Design layout in different devices

In responsive web design, adjusting the navigation content for visibility on a small device is crucial to provide an optimal user experience. This involves modifying the navigation layout and behavior to ensure easy access to navigation links on smaller screens.To achieve this, a media query is used to target the specific screen width where the adjustment should occur. Typically, the media query sets a maximum width, such as 600 pixels, which is commonly associated with small devices like mobile phones.Within the media query, the navigation content is adjusted using CSS styles. One common approach is to hide the regular navigation links and replace them with a more compact and mobile-friendly menu icon or button.

In the example provided, the navigation content is initially designed as a flex container (display: flex;) with the logo positioned to the left and an unordered list (ul) representing the menu positioned to the right. The menu itself is initially set to display: none;, effectively hidden.However, within the media query, the menu is set to display: block;, making it visible on small devices.

Meanwhile, the unordered list (ul) containing the navigation links is set to display: none;, effectively hiding it on small devices.

By implementing this approach, the navigation content is adjusted dynamically based on screen size. On small devices, the regular navigation links are hidden, and a more compact menu representation becomes visible. This ensures better visibility and usability of the navigation on smaller screens, enhancing the overall user experience.It's important to note that this is just one example of adjusting the navigation for small devices. The specific implementation may vary depending on design preferences and requirements.

```
<!DOCTYPE html>
<html>
<head>
<title>Computer Company - Responsive Design Layout Example</title>
<style>
body {
font-family: Arial, sans-serif;
margin: 0;
padding: 0;
   }

   .container {
max-width: 960px;
margin: 0 auto;
padding: 20px;
   }

   .header {
background-color: #333;
color: #fff;
padding: 20px;
text-align: center;
   }

   .content {
padding: 20px;
text-align: justify;
   }

   .footer {
background-color: #333;
color: #fff;
padding: 20px;
```

```
text-align: center;
  }

  @media screen and (max-width: 768px) {
    .container {
padding: 10px;
    }

    .header, .footer {
font-size: 16px;
    }
  }

  @media screen and (max-width: 480px) {
    .container {
padding: 5px;
    }

    .header, .footer {
font-size: 14px;
    }
  }
</style>
</head>
<body>
<div class="container">
<div class="header">
<h1>Computer Company</h1>
</div>

<div class="content">
<h2>Welcome to Computer Company!</h2>
<p>Computer Company is a leading provider of cutting-edge computer hardware and software solutions. We offer a wide range of products, including high-performance desktop computers, laptops, peripherals, and software applications.</p>
<p>Our team of experts is dedicated to delivering top-quality products that meet the needs of professionals, businesses, and enthusiasts alike. We strive to provide innovative solutions that enhance productivity, efficiency, and creativity.</p>
<p>Whether you're a gaming enthusiast, a business professional, or a student, Computer Company has the perfect technology solutions to empower your work and leisure. Experience the latest advancements in computer technology with us.</p>
</div>
```

```
<div class="footer">
<p>&copy; 2023 Computer Company. All rights reserved.</p>
</div>
</div>
</body>
</html>
```

In this example, the responsive design layout is customized for a computer-related company. The HTML structure includes a header, content, and footer sections, contained within a element with the class "container" to provide a maximum width of 960 pixels and center the layout on the page.

The CSS styles define the appearance of the elements, including background colors, padding, and text alignment. Media queries are used to adjust the layout for different screen sizes, with font sizes decreasing for smaller devices to enhance readability.

You can customize the content within the section to highlight the unique offerings, services, or products of your computer-related company. Additionally, feel free to modify the styling and add relevant images or interactive elements to create a visually appealing and engaging user experience.

Summary

- Prioritize content over layout by understanding user needs and goals. Deliver concise, well-structured content that engages users effectively.
- Implement responsive design techniques to ensure optimal viewing experiences across different devices and screen sizes. Use CSS media queries to adapt the layout and elements based on breakpoints.
- Utilize design and prototyping tools like Sketch, Figma, or Adobe XD to create mobile interfaces efficiently. Development frameworks such as React Native or Flutter help build responsive and cross-platform mobile applications.
- Focus on delivering a seamless user experience. Optimize navigation for mobile screens, ensure fast loading times, implement touch-friendly interactions, personalize content, and prioritize accessibility.
- Iterate and test designs with real users to refine the mobile experience further. Incorporate user feedback and usability testing to identify areas for improvement and enhance the overall user experience.

Keywords

- Content-first approach
- Responsive design
- CSS media queries
- Design and prototyping tools
- Development frameworks
- User experience (UX)

Self Assessment

1. Which approach emphasizes organizing content effectively for mobile experiences?
    A. Layout before content
    B. Content before layout
    C. Design before content
    D. Content alongside layout

2. What is the purpose of responsive design in mobile development?
    A. Optimizing content for desktop devices only
    B. Adapting layouts to different screen sizes and devices
    C. Creating fixed-width designs for specific devices
    D. Ignoring the need for mobile optimization

3. Which CSS feature allows you to define different styles based on screen sizes?
    A. Flexbox
    B. Media queries
    C. Grid layout
    D. Float positioning

4. Which of the following is a popular design and prototyping tool for mobile interfaces?
    A. Photoshop
    B. Illustrator
    C. Sketch
    D. InDesign

5. Which development frameworks enable building cross-platform mobile applications?
    A. AngularJS
    B. React Native
    C. jQuery Mobile
    D. Django

6. What does user experience (UX) focus on in mobile and beyond?
    A. Optimizing loading times
    B. Enhancing visual aesthetics
    C. Delivering a seamless interaction
    D. Increasing website traffic

7. What is an essential consideration for mobile navigation optimization?
    A. Increasing the number of navigation links
    B. Using complex gestures for interaction
    C. Minimizing the number of clicks required
    D. Displaying navigation menus in a sidebar

8. Why is accessibility important in mobile design?
    A. It helps increase website rankings in search engines.
    B. It ensures compatibility with various web browsers.
    C. It improves user engagement and satisfaction.
    D. It allows for faster loading times.

9. What does iterative design involve in mobile development?
    A. Creating a static design without user feedback
    B. Implementing user feedback to refine the design
    C. Skipping the testing phase for quicker development
    D. Making design decisions based on personal preferences

10. What is the purpose of usability testing in mobile design?
    A. Identifying areas for improvement in the user experience
    B. Enhancing the visual appeal of the mobile interface
    C. Increasing the speed and performance of the mobile app
    D. Providing a detailed analysis of coding techniques

Answers for Self Assessment

1 B 2 B 3 B 4 C 5 B 6 C 7 C 8 C

9 B 10 A

Review Questions

1. What is the main principle behind content before layout in mobile design?
2. What is responsive design and why is it important for mobile experiences?
3. How do media queries contribute to responsive design?
4. Name one popular design and prototyping tool used for mobile interfaces.
5. Which development frameworks facilitate building cross-platform mobile applications?
6. What does user experience (UX) encompass in the context of mobile and beyond?
7. Why is navigation optimization crucial for mobile interfaces?
8. What is the significance of accessibility in mobile design?
9. Explain the concept of iterative design in the context of mobile development.
10. What is the purpose of usability testing in mobile design?

Further Readings

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Multiple-column_Layout

Unit 13:Creating responsive websites to improve performance

Objectives

Understand the importance of prioritizing content before layout in mobile design to deliver effective and engaging user experiences.

Comprehend the principles and techniques of responsive design and its significance in ensuring optimal viewing experiences across various devices and screen sizes.

Identify and utilize appropriate design and prototyping tools for creating mobile interfaces, enabling efficient and streamlined development processes.

Recognize the key elements of user experience (UX) in the context of mobile and beyond, focusing on navigation optimization, accessibility, and iterative design for enhanced user satisfaction.

Introduction

Performance is an essential part of web design. A slow-loading website will frustrate users and lead to high bounce rates. There are two main aspects of web performance: loading and rendering. Loading refers to the time it takes for the web page's resources, such as images, CSS, and JavaScript, to download from the server. Rendering refers to the time it takes for the browser to display the web page on the user's screen.

There are a number of factors that can affect a website's performance, including the size and number of resources on the page, the type of resources, the browser being used, and the user's internet connection speed. There are a number of tools available to measure a website's performance, such as Google PageSpeed Insights, GTmetrix, and WebPageTest. By following tips such as optimizing images, using minified and compressed JavaScript and CSS files, using a CDN, and using lazy loading, you can improve your website's performance and give your users a better experience.

13.1Performance as design in responsive websites

Performance is a critical aspect of web design that directly impacts user experience, engagement, and overall success of a website. In the digital age where attention spans are short and competition is fierce, optimizing the performance of web pages has become an essential part of the design process. "Performance as Design" refers to the philosophy of considering performance factors as integral components of the web design strategy, rather than an afterthought. It involves prioritizing speed, responsiveness, and efficiency from the outset of the design process, ensuring that the end product delivers a seamless and satisfying user experience.

Web Pages Loading and Rendering:

Web pages loading and rendering are fundamental steps in delivering content to users. When a user enters a web address or clicks on a link, the browser retrieves the necessary files (HTML, CSS, JavaScript, images, etc.) from the server and starts rendering the page on the user's screen. Several factors influence this process, and the time it takes for a web page to load and render can significantly impact user satisfaction.

Loading speed is affected by various elements, such as the size and complexity of the page's resources, server response time, and the user's internet connection. Rendering speed, on the other hand, depends on how the browser processes and displays the retrieved content, as well as the efficiency of the underlying rendering engine.

Measuring Performance:

To ensure optimal performance, web developers and designers use various metrics to measure and analyze a web page's loading and rendering efficiency. Some common performance metrics include:

Page Load Time: This measures the time it takes for a web page to fully load and become usable. It includes the time taken to download all resources like HTML, CSS, JavaScript, images, and other media.

Time to First Byte (TTFB): This metric represents the time it takes for the server to respond to a user's request and start sending data back to the browser. A lower TTFB indicates a faster server response time.

First Contentful Paint (FCP): FCP measures the time it takes for the first piece of content to appear on the screen. It gives users a sense that the page is loading and is particularly crucial for perceived performance.

Largest Contentful Paint (LCP): LCP identifies the time it takes for the largest content element (image, video, etc.) to load and become visible. It reflects the visual stability of the page.

Time to Interactive (TTI): TTI gauges the time it takes for a web page to become fully interactive, allowing users to interact with buttons, forms, and other elements.

Cumulative Layout Shift (CLS): CLS measures the visual stability of a page by calculating unexpected layout shifts that might occur during the loading process.

By monitoring and optimizing these performance metrics, designers and developers can create web pages that load quickly, render smoothly, and provide users with a delightful experience. Prioritizing performance as a design principle ensures that users can access information efficiently, reducing bounce rates and enhancing the overall impact and success of a website.

13.2Responsive Design web pages loading and rendering

Responsive design is a web design approach that makes websites render well on a variety of devices, from desktops to mobile phones. This is achieved by using flexible layouts and media queries to adapt the website's content and structure to the user's screen size. One of the challenges of responsive design is ensuring that the website loads quickly on all devices. This is because the larger the screen size, the more resources are required to load the website.

There are a number of factors that can affect the loading speed of a responsive website, including:

- The size and number of images on the page
- The use of JavaScript and CSS
- The performance of the server
- The user's internet connection speed

There are a number of things that can be done to improve the loading speed of a responsive website, including:

Optimize images: Images are often the largest resources on a web page, so it's important to optimize them for size and compression. This can be done using a variety of tools, such as ImageOptim or TinyPNG.

Use minified and compressed JavaScript and CSS files: Minifying and compressing JavaScript and CSS files can help to reduce their size, which will improve loading times. This can be done using a variety of tools, such as Google's Closure Compiler or YUI Compressor.

Use a CDN: A CDN (content delivery network) can help to improve performance by delivering your website's resources from servers that are closer to your users. This can help to reduce latency and improve loading times.

Use lazy loading: Lazy loading is a technique that can be used to defer the loading of images and other resources until they are needed. This can help to improve loading times by only loading the resources that are actually visible on the page.

Use a caching plugin: A caching plugin can help to store static resources, such as images and CSS files, in the user's browser. This can help to improve loading times by reducing the number of requests that need to be made to the server.

Test your website on different devices: It's important to test your website on different devices to ensure that it performs well on all devices. This includes devices with different screen sizes, resolutions, and operating systems.

In addition to the technical aspects of loading speed, it's also important to consider the user experience (UX) implications of loading speed. A slow-loading website can be frustrating for users, and it can also lead to negative perceptions of the website's brand. Therefore, it's important to design responsive websites with loading speed in mind. This means using techniques such as lazy loading and minification to reduce loading times, and using clear and concise visual design to make the website easy to scan and navigate.

```
<!DOCTYPE html>
<html>
<head>
<title>Responsive Design Example</title>
<style>
    body {
        font-family: Arial, sans-serif;
        line-height: 1.6;
        margin: 0;
        padding: 0;
    }
    header {
        background-color: #333;
        color: #fff;
        padding: 10px;
        text-align: center;
    }
    .container {
        width: 90%;
        max-width: 1200px;
        margin: 0 auto;
        padding: 20px;
```

```
        display: flex;
        flex-wrap: wrap;
    }
    .column {
        flex: 1;
        min-width: 300px;
        padding: 10px;
    }
    @media screen and (max-width: 600px) {
        .column {
            flex-basis: 100%;
            margin-bottom: 20px;
        }
    }
img {
        max-width: 100%;
        height: auto;
        display: block;
        margin-bottom: 10px;
    }
</style>
</head>
<body>
<header>
<h1>Responsive Design </h1>
</header>
<div class="container">
<div class="column">
<imgsrc="c1.jpg" alt="Image 1">
<p align="justify">          Responsive design is a web design approach that makes websites render well on a variety of devices, from desktops to mobile phones. This is achieved by using flexible layouts and media queries to adapt the website's content and structure to the user's screen size. </p>
</div>
<div class="column">
<imgsrc="c2.png" alt="Image 2">
<p align="justify">One of the challenges of responsive design is ensuring that the website loads quickly on all devices. This is because the larger the screen size, the more resources are required to load the website.</p>
</div>
</div>
</body>
```

```
</html>
```

In this example, we have a simple responsive layout with two columns inside a flex container. The CSS uses media queries to adjust the layout for screens with a maximum width of 600 pixels or smaller, where the columns will stack vertically. Images are set to scale within their container to ensure they adapt to different screen sizes.

13.3How to measure the performance of website?

There are a number of ways to measure the performance of a website. Some of the most common metrics include page load time, time to first byte, number of requests, size of resources, bounce rate, and average session time. There are a number of tools that can be used to measure the performance of a website, such as Google PageSpeed Insights, GTmetrix, and WebPageTest.

By measuring the performance of your website, you can identify areas where it can be improved. Once you have identified these areas, you can make changes to your website's code or hosting environment to improve its performance.

There are a number of ways to measure the performance of a website. Some of the most common metrics include:

Page load time: This is the time it takes for a web page to load completely in the user's browser.

Time to first byte (TTFB): This is the time it takes for the first byte of data to be transferred from the server to the user's browser.

Number of requests: This is the number of requests that are made to the server to load the web page.

Size of resources: This is the size of the images, CSS, and JavaScript files that are loaded to render the web page.

Bounce rate: This is the percentage of users who leave the website after viewing only one page.

Average session time: This is the average amount of time that users spend on the website.

There are a number of tools that can be used to measure the performance of a website. Some of the most popular tools include:

Google PageSpeed Insights: This is a free tool that provides a score for the performance of a website.

GTmetrix: This is a paid tool that provides a more detailed analysis of the performance of a website.

WebPageTest: This is a free tool that allows you to test the performance of a website from different locations.

By measuring the performance of your website, you can identify areas where it can be improved. Once you have identified these areas, you can make changes to your website's code or hosting environment to improve its performance.

Optimize images: Images are often the largest resources on a web page, so it's important to optimize them for size and compression.

Use minified and compressed JavaScript and CSS files: Minifying and compressing JavaScript and CSS files can help to reduce their size, which will improve loading times.

Use a CDN: A CDN (content delivery network) can help to improve performance by delivering your website's resources from servers that are closer to your users.

Use lazy loading: Lazy loading is a technique that can be used to defer the loading of images and other resources until they are needed. This can help to improve loading times by only loading the resources that are actually visible on the page.

Use a caching plugin: A caching plugin can help to store static resources, such as images and CSS files, in the user's browser. This can help to improve loading times by reducing the number of requests that need to be made to the server.

Here are various tools and metrics available to measure website performance. Here are some common methods to help you evaluate the performance of a website:

Page Load Time: Page load time is one of the most critical metrics to assess website performance. It measures the time taken for a web page to fully load and become usable to the user. You can use various online tools such as Google PageSpeed Insights, WebPageTest, or GTmetrix to measure page load time for your website.

Time to First Byte (TTFB): TTFB measures the time it takes for the server to respond to a user's request and start sending data back to the browser. A lower TTFB indicates a faster server response time, which contributes to faster overall page loading. Tools like Pingdom and WebPageTest can provide TTFB measurements.

First Contentful Paint (FCP): FCP measures the time it takes for the first piece of content to appear on the screen. It is essential for providing users with feedback that the page is loading. Google PageSpeed Insights and Lighthouse report FCP data.

Largest Contentful Paint (LCP): LCP identifies the time it takes for the largest content element (image, video, etc.) to load and become visible. A fast LCP ensures that users see meaningful content quickly. Google PageSpeed Insights and Lighthouse also provide LCP measurements.

Total Blocking Time (TBT): TBT measures the total amount of time the page is unresponsive to user input while loading. It is an essential metric for assessing the page's interactivity during the loading process.

Cumulative Layout Shift (CLS): CLS measures the visual stability of a page by calculating unexpected layout shifts during the loading process. A low CLS value indicates that the page elements are not moving around unexpectedly.

Browser Developer Tools: Modern browsers have built-in developer tools that allow you to analyze network activity, performance timings, and other metrics. You can access these tools by pressing F12 or right-clicking on the page and selecting "Inspect" or "Inspect Element."

Real User Monitoring (RUM): RUM tools track actual user interactions on your website and provide valuable data on performance from different locations and devices.

Web Server Logs: Server logs can help you analyze server response times, request rates, and other server-related performance metrics.Google Analytics: Google Analytics can provide insights into website performance, including page load times and user behaviour.

```
<!DOCTYPE html>
<html>
<head>
<title>Traveling Company - Responsive Web Design</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    /* Reset some default styles */
    body, h1, p {
```

```
        margin: 0;
        padding: 0;
    }

    body {
        font-family: Arial, sans-serif;
        line-height: 1.6;
    }

    header {
        background-color: #333;
        color: #fff;
        padding: 10px;
        text-align: center;
    }

    nav {
        background-color: #444;
        color: #fff;
                                text-align: center;
        padding: 10px;
    }

    .container {
        width: 90%;
        max-width: 1200px;
        margin: 0 auto;
        padding: 20px;
        display: flex;
        flex-wrap: wrap;
    }

    .column {
        flex: 1;
        min-width: 300px;
        padding: 10px;
    }

img {
        max-width: 100%;
```

```
        height: auto;
        display: block;
        margin-bottom: 10px;
    }

    footer {
        background-color: #333;
        color: #fff;
        text-align: center;
        padding: 10px;
    }

    /* Media query for tablets and larger screens */
    @media screen and (min-width: 768px) {
        .column {
            flex-basis: 50%;
        }
    }

    /* Media query for smaller screens (e.g., smartphones) */
    @media screen and (max-width: 600px) {
        nav {
            display: block;
        }
        .column {
            flex-basis: 100%;
            margin-bottom: 20px;
        }
    }
</style>
</head>
<body>
<header>
<h1>Traveling Company</h1>
</header>
<nav>
    Home |
    Destinations |
    Tours |
    Contact
```

```
</nav>
<div class="container">
<div class="column">
<imgsrc="tapi.jpg" alt="Destination 1">
<h2>Taipei Palace Museum</h2>
<p>Taipei is the heart of Taiwan, featuring convenient transportation and a number of attractions that make it a first stop for visitors to Taiwan. Taipei 101 and Taipei National Palace Museum are two of Taipei's landmark attractions. From the top of 101, visitors get a gorgeous night view of the entire city, while a visit to the National Palace Museum offers a profound experience of Chinese traditional culture.</p>
</div>
<div class="column">
<imgsrc="tapi1.jpg" alt="Destination 2">
<h2>TAIPEI </h2>
<p>Taipei 101 and Taipei National Palace Museum are two of Taipei's landmark attractions. From the top of 101, visitors get a gorgeous night view of the entire city, while a visit to the National Palace Museum offers a profound experience of Chinese traditional culture. Taipei is also famous for the gourmet snacks available in its dazzling night markets, each with their own special characteristics. Among them, Yongkang Street is a veritable paradise for foodies.</p>
</div>
</div>
<footer>
&copy; 2023 Traveling Company. All rights reserved.
</footer>
</body>
</html>
```

In the above code, the layout adjusts according to the screen size. The navigation menu collapses into a single column on smaller screens, and the destination columns stack vertically, providing a user-friendly experience on devices of different sizes.

Summary

- Performance as design prioritizes speed and efficiency, optimizing loading and rendering for seamless user experience.
- Measuring metrics like load time, TTFB, FCP, and LCP drives continuous improvement in responsive web design.
- Efficient resource utilization and responsive layouts contribute to enhanced web page loading and rendering.
- Regular performance monitoring ensures a smooth user experience across different devices and screen sizes.

Keywords

- Rendering
- Responsive
- Metrics
- Load Time

- TTFB (Time to First Byte)
- FCP (First Contentful Paint)
- LCP (Largest Contentful Paint)
- CLS (Cumulative Layout Shift)

Self Assessment

1. Which design principle focuses on prioritizing speed and efficiency in web development?
   A. User Experience (UX)
   B. Performance as Design
   C. Responsive Design
   D. Visual Aesthetics

2. What is the metric that measures the time it takes for a web page to fully load and become usable?
   A. Time to First Byte (TTFB)
   B. First Contentful Paint (FCP)
   C. Largest Contentful Paint (LCP)
   D. Page Load Time

3. Responsive design aims to create web pages that:
   A. Load faster on desktops
   B. Have more animations and effects
   C. Adapt to different devices and screen sizes
   D. Are compatible only with the latest browsers

4. Which metric measures the time it takes for the server to respond to a user's request and start sending data back to the browser?
   A. Time to First Byte (TTFB)
   B. First Contentful Paint (FCP)
   C. Largest Contentful Paint (LCP)
   D. Cumulative Layout Shift (CLS)

5. Performance measurement tools like Google PageSpeed Insights and GTmetrix provide insights into:
   A. User engagement metrics
   B. Server response time
   C. JavaScript code analysis
   D. Web page loading and rendering performance

6. What does LCP stand for in performance measurement metrics?
   A. Load Time for Content
   B. Largest Contentful Paint
   C. Last Contentful Paint
   D. Longest Central Point

7. Which technique helps in delivering smaller image sizes based on the user's device capabilities?
    A. Adaptive Images
    B. Dynamic Images
    C. Progressive Images
    D. Responsive Images

8. What is the main advantage of using a mobile-first approach in responsive design?
    A. It makes the website look more modern.
    B. It improves search engine ranking.
    C. It optimizes the website for mobile users with limited resources.
    D. It enhances the loading speed on all devices.

A. Which tool allows developers to analyze network activity, performance timings, and other metrics?
    A. WebPageTest
    B. Lighthouse
    C. Google PageSpeed Insights
    D. Browser Developer Tools

9. Continuous performance monitoring helps ensure:
    A. Consistent page load times on all devices
    B. Quick fixes for all website issues
    C. Improved user engagement and conversion rates
    D. No need for further website updates

Answers for Self Assessment

1 B 2 D 3 C 4 A 5 D 6 B 7 A 8 C

9 D 10 C

Review Questions

1. What does "Performance as Design" refer to in web development?
2. Name two metrics used to measure web page loading time.
3. What is the primary goal of responsive design?
4. What does TTFB stand for, and what does it measure?
5. How can adaptive images improve web page loading performance?
6. Name one tool that provides insights into web page loading and rendering performance.
7. Why is the mobile-first approach important in responsive design?
8. What does LCP stand for, and what does it indicate in performance measurement?

9. How does continuous performance monitoring benefit a website?
10. Mention one technique to optimize web page rendering on different devices.

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

Aseem Khanna, Lovely Professional University

Unit 14:Creating responsive websites to improve performance

Objectives

Creating responsive websites to ensure a seamless user experience across devices and faster loading times, improving overall performance.

Increase mobile traffic engagement and enhance search engine rankings by optimizing for mobile-friendliness through responsive design.

Introduction

The performance of a responsive website is important for both the user experience and the search engine ranking. A slow-loading website will frustrate users and make them more likely to leave, while a fast-loading website will keep users engaged and improve your chances of ranking higher in search results.

One of the best ways to improve the performance of a responsive website is to clean up the code. This means removing any unnecessary code, optimizing images, and minifying and gzipping the code. By cleaning up the code, you can significantly reduce the file size of your website, which will improve loading times.

14.1Improve performanceby cleaning up your code

A responsive website's speed may be greatly enhanced by eliminating unnecessary code. The code must be optimized by deleting unused lines, compressing unneeded data, and enhancing picture quality. Reducing the file size of your website by cleaning up the code may drastically improve loading speeds.

Benefits of cleaning up code

There are several benefits to cleaning up code for responsive websites. These include:

Improved performance: As mentioned above, cleaning up code can significantly reduce the file size of your website, which will improve loading times. This will provide a better user experience for your visitors and make your website more likely to rank higher in search results.

Reduced maintenance costs: Clean code is easier to maintain than cluttered code. This means that you will spend less time fixing bugs and updating your website.

Enhanced security: Clean code is less likely to contain security vulnerabilities. This can help to protect your website from hackers and malware.

How to clean up code

There are a few different ways to clean up code for responsive websites. One way is to use a code minifier. A code minifier will remove unnecessary whitespace and comments from your code, which will reduce the file size. Another way to clean up code is to use a code optimizer. A code optimizer will analyze your code and identify areas where it can be optimized for performance.

Enhancing performance is crucial for any responsive website, and one effective strategy is to clean up your code. A responsive website adjusts seamlessly across devices, ensuring an optimal user experience. However, bloated or disorganized code can lead to slower loading times and a less efficient design. By diligently tidying up the code, developers can streamline the website's performance, reduce load times, and deliver a more responsive and user-friendly experience. This approach empowers the website to deliver its features effectively while maintaining a high level of performance and user satisfaction.

Improving performance by cleaning up code for responsive websites can involve several types of code cleanup and optimization techniques:

Remove Unused Code: Eliminate redundant or unused HTML, CSS, and JavaScript code to reduce the overall file size and improve loading speed.

Minification and Compression: Minify and compress CSS, JavaScript, and HTML files to reduce their size and decrease load times.

Optimize Images: Compress images without compromising quality, use responsive image techniques, and implement lazy loading to reduce image loading times.

Reduce HTTP Requests: Combine CSS and JavaScript files and use CSS sprites to minimize the number of HTTP requests required to load the page.

Eliminate Render-Blocking Resources: Optimize CSS and JavaScript loading to prevent render-blocking and improve initial page rendering.

Cache Management: Utilize browser caching and server-side caching mechanisms to store static resources, reducing server requests and improving page load times for returning visitors.

Use Responsive Design Frameworks: Implement responsive design frameworks like Bootstrap or Foundation to build mobile-first and adaptive layouts, ensuring a consistent user experience across devices.

Media Query Optimization: Use efficient media queries and breakpoints to target specific devices and reduce unnecessary CSS styles for other screen sizes.

Optimize Font Usage: Limit the number of fonts and font weights used on the website to reduce load times.

Code Refactoring: Review and refactor code to improve its efficiency, readability, and maintainability.

14.2Improve performance and minimizing http requests

Improving website performance and minimizing HTTP requests are fundamental strategies in web development that significantly impact a website's speed and user experience. HTTP requests occur when a web browser fetches resources, such as HTML, CSS, JavaScript, images, and other assets, from the server to display a webpage. Reducing the number of HTTP requests helps to streamline the loading process, leading to faster page load times and improved overall performance. By optimizing resource usage, combining files, and leveraging caching, developers can effectively minimize HTTP requests and create a more efficient and responsive website, enhancing user satisfaction and engagement. This introduction sets the stage for exploring the benefits and techniques of minimizing HTTP requests to achieve optimal website performance.

HTTP requests are the building blocks of the web. Every time you visit a website, your browser sends a series of HTTP requests to the server that hosts the website. The server then responds to these requests by sending back the content of the website.

The number of HTTP requests that a website makes can have a significant impact on its performance. Each HTTP request takes time, so the more requests that a website makes, the longer it will take to load.There are a number of ways to minimize the number of HTTP requests that a

website makes. One way is to combine multiple resources into a single file. For example, you could combine all of the CSS files for a website into a single CSS file. This would reduce the number of HTTP requests from three to one.

Another way to minimize HTTP requests is to use lazy loading. Lazy loading is a technique where images and other resources are only loaded when they are needed. This can help to improve the performance of websites with a lot of images.

Benefits of minimizing HTTP requests

There are several benefits to minimizing HTTP requests. These include:

Improved performance: As mentioned above, minimizing HTTP requests can improve the performance of websites by reducing the loading time. This will provide a better user experience for your visitors.

Reduced bandwidth usage: Minimizing HTTP requests can also reduce the bandwidth usage of websites. This can save you money on your hosting costs.

Enhanced SEO: Google and other search engines take into account the performance of websites when ranking them in search results. This means that minimizing HTTP requests can help to improve your website's search engine ranking.

How to minimize HTTP requests

There are a number of ways to minimize HTTP requests. Some of the most common methods include:

- Combining resources: Combining multiple resources into a single file can reduce the number of HTTP requests.
- Using lazy loading: Lazy loading is a technique where images and other resources are only loaded when they are needed. This can help to improve the performance of websites with a lot of images.
- Using a content delivery network (CDN): A CDN can help to improve the performance of websites by caching static content on servers that are closer to your visitors.

Combining resources

Here is an example of how to combine multiple resources into a single file:

```
<link rel="stylesheet" href="style.css">
<link rel="stylesheet" href="style2.css">
<link rel="stylesheet" href="style3.css">
```

Instead of having three separate CSS files, we can combine them into a single file called `style.css`. This would reduce the number of HTTP requests from three to one.

Using lazy loading

```
<imgsrc="image.jpg" data-src="image.jpg" alt="Image">
```

The data-src attribute is used to specify the URL of the image that will be loaded when the user scrolls to the image. This means that the image will not be loaded until it is needed, which can help to improve the performance of websites with a lot of images.

Using a content delivery network (CDN)

```
<imgsrc="https://mycdn.com/image.jpg" alt="Image">
```

Instead of loading the image from the website's server, we can load it from a CDN. A CDN is a network of servers that are located all over the world. This means that images can be loaded from the server that is closest to the user, which can improve the performance of the website.

14.3What is conditionally loading content and reflows and repaints

Conditional loading is a technique used to load content only when it is needed. This can be used to improve the performance of websites by reducing the amount of data that needs to be loaded. There are a number of ways to conditionally load content. One common method is to use media queries. Media queries allow you to specify different CSS for different screen sizes. This means that you can load different content for different devices, such as loading a smaller image for mobile devices.Another way to conditionally load content is to use JavaScript. JavaScript can be used to check for certain conditions, such as the user's bandwidth or the user's preferences. If the condition is met, then the content will be loaded.Conditional loading can be a powerful tool for improving the performance of websites. By loading only the content that is needed, you can reduce the loading time of your website and improve the user experience.

some examples of conditional loading as under:

- To improve the performance of websites with a lot of images. By loading images only when they are needed, you can reduce the loading time of your website.
- To improve the user experience for users with slow internet connections. By loading only the essential content for users with slow internet connections, you can improve the user experience and prevent users from getting frustrated.
- To improve the accessibility of websites. By loading content only when it is needed, you can improve the accessibility of your website for users with disabilities.

```
<!DOCTYPE html>
<html>
<head>
<title>Conditional Content Loading Example</title>
<style>
    /* CSS to demonstrate reflows and repaints */
    .content {
background-color: lightblue;
width: 200px;
height: 200px;
    }
</style>
<script>
    // Conditionally load content and manage reflows and repaints
functionloadAdditionalContent() {
        // Check if the additional content is already loaded
if (!document.getElementById("additionalContent")) {
            // Create a new element for additional content
varadditionalContent = document.createElement("div");
            additionalContent.id = "additionalContent";
additionalContent.innerHTML = "Additional Content Loaded!";

            // Append the additional content to the page
document.getElementById("mainContent").appendChild(additionalContent);
```

```
        }
    }
</script>
</head>
<body>
<h1>Conditional Content Loading Example</h1>

<button onclick="loadAdditionalContent()">Load Additional Content</button>

<div class="content" id="mainContent">
<!-- Initial content here -->
</div>
</body>
</html>
```

This HTML example showcases the concept of conditionally loading content and effectively managing reflows and repaints in web development. The HTML document begins with the declaration of the HTML5 document type. The <head> section contains metadata, such as the title of the web page, and includes a <style> element defining CSS styles for the class "content." The CSS styles set a light blue background color and a fixed width and height of 200 pixels for elements with this class.

Within the <head>, a <script> element embeds JavaScript code responsible for conditionally loading content. The JavaScript function named loadAdditionalContent() is called when the button is clicked. It checks if an element with the id "additionalContent" already exists on the page using document.getElementById("additionalContent"). If the element does not exist, it means the additional content has not been loaded yet.

Upon verifying that the content is not present, the JavaScript code dynamically creates a new element to hold the additional content. The new element is assigned the id "additionalContent" and its innerHTML is set to display the text "Additional Content Loaded!".

To complete the conditional loading process, the JavaScript code appends the newly created element with the additional content as a child of the element with the id "mainContent". This ensures that the additional content appears within the specified container on the web page.

Throughout this process, the code effectively manages reflows and repaints. When content is conditionally loaded, the browser calculates and updates element positions and dimensions in the Document Object Model (DOM). However, by minimizing unnecessary changes to the DOM and avoiding frequent style updates, the number of reflows and repaints is reduced, leading to better rendering performance and a smoother user experience.

```
<!DOCTYPE html>
<html>
<head>
<title>Conditional Loading</title>
</head>
<body>
<div id="content">
<h1>This is the default content.</h1>
</div>
```

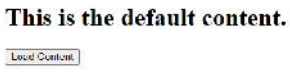
```
<script>
functionloadContent() {
varxhr = new XMLHttpRequest();
xhr.open("GET", "content.html", true);
xhr.onload = function() {
if (xhr.status === 200) {
var content = xhr.responseText;
document.getElementById("content").innerHTML = content;
  }
 };
xhr.send();
}
</script>
<button onclick="loadContent()">Load Content</button>
</body>
</html>
```

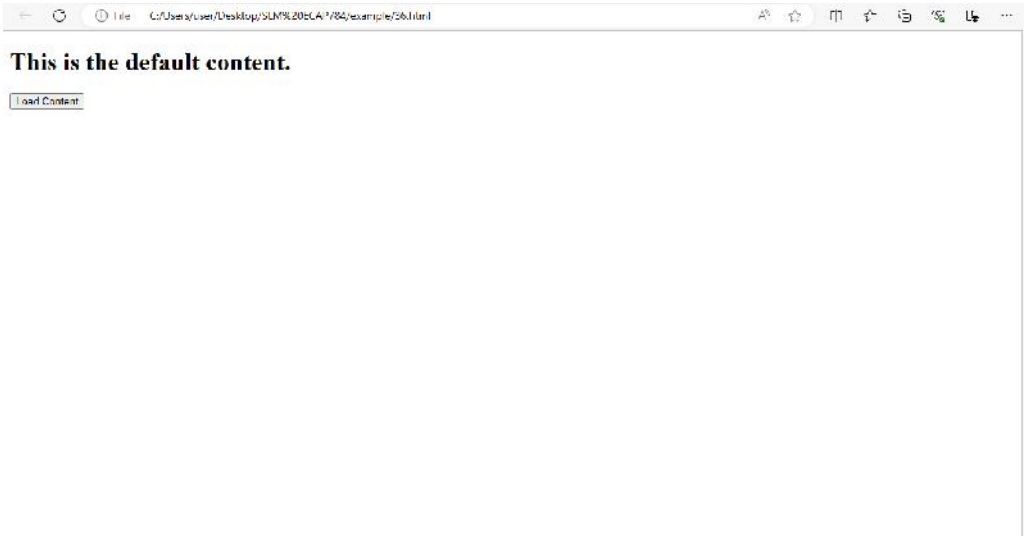
In this example, we have a simple web page with a button and a element with the class "content" serving as the main content container. When the button is clicked, the loadAdditionalContent() function is called. This function checks if the additional content (identified by the "additionalContent" id) is already present on the page. If not, it creates a new element containing the text "Additional Content Loaded!" and appends it to the "mainContent" . This approach ensures that the additional content is loaded only once when the button is clicked.

In this example, the default content is a heading that says "This is the default content." When the user clicks the "Load Content" button, the JavaScript code will load the content of the content.html file and replace the default content with the new content.

The content.html file could contain any content, such as text, images, or videos. When the new content is loaded, the browser will need to perform a reflow and repaint. The reflow is necessary because the new content may have a different size or position than the default content. The repaint is necessary because the new content may have different colors or styles.

The reflow and repaint can be expensive operations, so it is important to use conditional loading to only load content that is actually needed. In this example, the "Load Content" button is used to trigger the loading of the new content. This ensures that the reflow and repaint are only performed when the user actually wants to see the new content.





Summary

- Conditional loading is a technique for loading content only when it is needed.
- Reflows and repaints are two operations that the browser performs when the layout of a web page changes.
- Conditional loading can be used to reduce the number of reflows and repaints that occur on a web page.
- To minimize reflows and repaints, developers should avoid frequent style changes, use efficient CSS and HTML structure, and employ techniques like batching DOM updates.

Keywords

- Conditionally loading content
- Reflows

- Repaints
- Optimizing web page performance
- Dynamic content
- Page load times
- User experience
- On-demand loading
- Efficient CSS and HTML
- Minimizing reflows and repaints

Self Assessment

1. When is conditionally loading content beneficial?
    A. Always, regardless of page complexity
    B. When the web page has dynamic content that should load on-demand
    C. Only for small-sized web pages
    D. Only for static web pages

2. What are reflows in web development?
    A. The process of loading additional content conditionally
    B. The process of calculating and updating element positions and dimensions in the DOM
    C. The process of optimizing CSS styles
    D. The process of serving cached content to users

3. Which of the following is a consequence of frequent reflows and repaints on a web page?
    A. Faster page load times
    B. Smoother user interactions
    C. Improved rendering performance
    D. Slower user experience and page rendering

4. How can developers minimize reflows and repaints on a web page?
    A. Increasing the use of CSS animations
    B. Frequent DOM manipulations and style changes
    C. Using efficient CSS and HTML structure
    D. Using large images for better visual appeal

5. Which of the following can help optimize web page performance when dealing with dynamic content?
    A. Avoiding conditionally loading content
    B. Loading all content upfront, regardless of the user's interaction
    C. Conditionally loading additional resources when needed
    D. Reducing the use of JavaScript in web development

6. What is the main advantage of conditionally loading content on a web page?

A. Improved page layout
B. Faster initial page load time
C. Decreased reliance on server resources
D. Better user interactions with dynamic elements

7. Which technique can help improve the efficiency of CSS and HTML?
   A. Frequent style changes
   B. Inlining all CSS styles
   C. Using external CSS files
   D. Using a single, large CSS file for the entire website

8. What is the purpose of selectively executing scripts when conditionally loading content?
   A. To speed up the browser rendering process
   B. To ensure all scripts are executed simultaneously
   C. To reduce the risk of conflicts between different scripts
   D. To prioritize executing the largest script first

9. When should a web developer consider conditionally loading content?
   A. Only for static web pages
   B. For all types of web pages, regardless of content
   C. Only for web pages with minimal user interactions
   D. For web pages with dynamic content that loads on-demand

10. How does minimizing reflows and repaints impact the overall user experience?
    A. It has no significant impact on the user experience.
    B. It leads to faster page load times and smoother interactions.
    C. It increases the risk of rendering issues on the web page.
    D. It improves the loading time of large images and videos.

Answers for Self Assessment

1 B 2 B 3 D 4 C 5 C 6 B 7 C 8 C

9 D 10 B

Review Questions

1. What is the main purpose of conditionally loading content on a web page?
2. Define reflows in the context of web development.
3. How do frequent reflows and repaints affect web page performance?

4. Name one technique to minimize reflows and repaints on a web page.
5. When is conditionally loading additional resources most beneficial?
6. What advantage does selectively executing scripts offer when conditionally loading content?
7. How can developers optimize CSS and HTML for better web page performance?
8. In what scenarios should a web developer consider conditionally loading content?
9. What impact does minimize reflows and repaints have on user experience?
10. What is the purpose of using external CSS files in web development?

Further Readings

https://www.w3schools.com/css/css3_mediaqueries.asp

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in