**HTML/CSS Notes - Week 4**

Date:- 10/03/2022

**CSS Properties**

a) CSS box-sizing Property- The box-sizing property allows us to include the padding and border in an element's total width and height.

b) justify-content- This property is used to align the whole grid horizontally inside the container. (Mostly used with grid elements)

c) align-content- property is used to vertically align the whole grid inside the container. (Mostly used with grid elements)

1. Responsive Web Design(The Viewport) - The browser's viewport is the area of the window in which web content can be seen.

In HTML5, we can take control of the viewport using the <meta> tag which gives the browser instructions on how to control the page's dimensions and scaling. In every HTML page, we must include:-

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

a) The width=device-width part sets the width of the page to follow the screen-width of the device

b) The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

2. Responsive Web Design(Grid-View) - A grid-view helps us to divide our page into columns. A responsive grid-view has 12 columns, and has a total width of 100%, and will shrink and expand as we resize the browser window. Example:-

| <**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**>Responsive Grid View</**title**>   <**style**>  \* {  box-sizing: border-box;  }    .grid-items {  display: flex;  }    .nav-items {  width: 25%;  border: 2px solid #000;  }    .demo-content {  width: 75%;  border: 2px solid #000;  padding: 0 10px;  }    @media screen and (max-width: 800px) {  .grid-items {  display: block;  }  .nav-items {  width: 100%;  }  .demo-content {  width: 100%;  }  }  </**style**> </**head**>  <**body**>  <**div** class="grid-items">  <**div** class="nav-items col-1">  <**ul**>  <**li**>Item 1</**li**>  <**li**>Item 2</**li**>  <**li**>Item 3</**li**>  <**li**>Item 4</**li**>  <**li**>Item 5</**li**>  </**ul**>  </**div**>   <**div** class="demo-content col-2">  <**p**>  Lorem ipsum dolor, sit amet consectetur adipisicing elit. Earum obcaecati dolor accusantium sint esse numquam inventore! Consequatur voluptatem nostrum, distinctio et esse eos ea ex sapiente impedit dolorum earum porro?  </**p**>  </**div**>  </**div**> </**body**>  </**html**> |
| --- |

3. Responsive Web Design(Images)- In order to make an image responsive on every device, we make use of the properties:-

-> max-width: 100%;

-> height: auto;

By using these properties, the image will be responsive and scale down as per its size.

4. CSS Grid Layout- It is a grid-based layout system, with rows and columns, making it easier to design web pages. A grid layout consists of a parent element, with one or more child elements. In order to make an HTML element a grid container, display property is set to grid or inline-grid.

-> We can adjust the gap size by using one of the following properties:

a) **column-gap**:- It sets the gap between the columns

b) **row-gap**:- It sets the gap between the rows

c) **gap**:- It is a shorthand property for the row-gap and the column-gap properties

Some of the common grid-layout properties include:-

a) grid-template-columns- It defines the number of columns in your grid layout, and it can define the width of each column. If we're adding 4 columns, we can specify the width of the 4 columns, or "auto" if all columns should have the same width.

b) grid-template-rows- This property defines the height of each row.

c) grid-column- This property defines which column(s) to place an item. We define where the item will start, and where the item will end. The grid-column property is a shorthand property for the grid-column-start and the grid-column-end properties. To place an item, we can refer to line numbers. Example:-

| .item\_1 {  grid-column: 1 / 4; # Start "item1" on column 1 and end before column 4 } |
| --- |

d) grid-row- This property defines on which row(s) to place an item. We define where the item will start, and where the item will end. The grid-row property is a shorthand property for the grid-row-start and the grid-row-end properties. To place an item, we can refer to line numbers.

Example:-

| <**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**>Grid Layout Example</**title**>   <**style**>  .grid-container {  display: grid;  grid-template-columns: auto auto auto;  gap: 10px;  background-color: #3d2d70;  padding: 10px;  }    .grid-container>div {  background-color: rgba(255, 255, 255, 0.8);  text-align: center;  padding: 20px 0;  font-size: 20px;  }    .item\_1 {  grid-row: 1 / 4;  }    .item\_11 {  grid-column: 2 / 4;  }  </**style**> </**head**>  <**body**>  <**div** class="grid-container">  <**div** class="item\_1">1</**div**>  <**div** class="item\_2">2</**div**>  <**div** class="item\_3">3</**div**>  <**div** class="item\_4">4</**div**>  <**div** class="item\_5">5</**div**>  <**div** class="item\_6">6</**div**>  <**div** class="item\_7">7</**div**>  <**div** class="item\_8">8</**div**>  <**div** class="item\_9">9</**div**>  <**div** class="item\_10">10</**div**>  <**div** class="item\_11">11</**div**>  <**div** class="item\_12">12</**div**>  <**div** class="item\_13">13</**div**>  <**div** class="item\_14">14</**div**>  <**div** class="item\_15">15</**div**>  <**div** class="item\_16">16</**div**>  </**div**> </**body**>  </**html**> |
| --- |

5. CSS Flexbox Layout- The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning. In order to use the Flexbox model, we need to first define a flex container and give it the property of display: flex;. Some of the flex container properties include:-

a) **flex-direction**- This property defines in which direction the container will stack the flex items. This property can be given different types of values like:-

-> column- This value stacks the flex items vertically (from top to bottom).

-> column-reverse- This value stacks the flex items vertically (but from bottom to top)

-> row- This value stacks the flex items horizontally (from left to right)

-> row-reverse- This value stacks the flex items horizontally (but from right to left)

b) **flex-wrap**- This property specifies whether the flex items should wrap or not. This property can be given different types of values like:-

-> wrap- This value specifies that the flex items will wrap if necessary.

-> wrap-reverse- This value specifies that the flexible items will wrap if necessary, in reverse order.

c) **flex-flow**- This is a shorthand property for setting both the flex-direction and flex-wrap properties.

We can also make use of CSS properties like "justify-content" and "align content" to perfectly center the flex items.

d) **align-self**- This property specifies the alignment for the selected item inside the flexible container.

e) **flex-grow**- This property specifies how much a flex item will grow relatively to the rest of the flex items. Default value is 0.

f) **flex-shrink**- This property specifies how much a flex item will shrink relative to the rest of the flex items.

Example:-

| <**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**>Flexbox Layout Example</**title**>   <**style**>  \* {  box-sizing: border-box;  }    .flex-container {  display: flex;  flex-direction: row-reverse;  height: 200px;  background-color: rgb(216, 219, 38);  }    .flex-container>div {  background-color: #f1f1f1;  margin: 10px;  padding: 20px;  font-size: 30px;  }  </**style**> </**head**>  <**body**>  <**div** class="flex-container">  <**div** style="flex-grow: 5">1</**div**>  <**div** style="align-self: center">2</**div**>  <**div**>3</**div**>  <**div** style="align-self: flex-start">4</**div**>  <**div**>5</**div**>  <**div** style="align-self: flex-end">6</**div**>  </**div**> </**body**>  </**html**> |
| --- |

CSS Variables- These are just like simple variables of any other programming language. These variables are used to store values and have a scope in which the variables can be used. A variable is defined by using two dashes(–) at the beginning and then the name which is case-sensitive. The benefit of variables is that it allows the same values to be reused at multiple places and updated/modified from one place.

Example:-

item\_1 {   
 --body-color: blue;   
}

The "item\_1" above indicates the selector that specifies the scope of the custom property. If we define the custom properties on the :root pseudo-class, then it will be globally available to our HTML document.

-> The var() function in CSS is used to insert the custom property value. The name of the variable can be passed as the argument to the var() function. Syntax:-

var(--property-name, value)

-> "value" here is an optional parameter in which we specify a value that works as a substitution in case no value is provided for the custom property.