Öğrendiklerim:

HTML:

Quite simply, **HTML** (Hypertext Markup Language) is used to create the actual content of the page, such as written text, and **CSS** (Cascade Styling Sheets) is responsible for the design or style of the website, including the layout, visual effects and background color.

Commnet: <!-- ... -->

<img src = “ “ alt = “ “ width = height = > (self closing)

<a href = “ “> </a>

<p> </p>

<main> </main>

Nested statements

<ul> </ul> unordered lists nested with <li> </li>

<ol> </ol> ordered lists nested with <li> </li>

<input type = “text”> (getting input from user, self closing) ( creates a box for user to enter smth), we can use placeholders (writings in that box before user enters smth by nesting)

<input type = “text” placeholder = “enter something!” >

* İnput type = “text” can be used for forms and form answers are directed to the “action” url. For example <form action = “ URL”> <input type = “text” placeholder = “type smth” > </form>. If we want this input to be required to fill, we use <input type = “text” required> then it will be required to fill.
* İnside form tag, we can use button type = “Submit” to create a submit button for our input box. <button type = “submit”> Submit (this will appear on button) </button>

For question with only 1 answer we use radio buttons. If you click a radio button, all the other buttons with the same name will be deselected. Radio buttons are used in <label> </label>.

It is a good practice to use for attribute with id’s of buttons that they are bounded.

<label for = “indoor”>

Indoor <input id = “indoor” type = “radio” name = “indoor-outdoor”>

</label>

<label for = “outdoor”>

Outdoor <input id = “outdoor” type = “radio” name = “indoor-outdoor”>

</label>

You can also use this structure inside forms.

Checkbox has the same structure with input radio however it allows multiple answers.

Radio and checkbox inputs give their form values to the url by their value = “ “. İf it is omitted then it is reported as name + “on” if checked. İf not for example indoor-outdoor (name) + indoor (value) is given. You still need to give a name to the radio check outside of input.

Outdoor <input id = “outdoor” type = “radio” name = “indoor-outdoor” value = “outdoor”>

We can also by default check all boxes of radio and checked boxes using “checked” statement.

<input type = “radio” name = “test-name” checked>

We use <div> </div> to seperate some parts.

We need to tell browser which HTML version we are using at the beginning of the document since html is being updated regularly

So a proper structure for our codes is the fallowing :

<!DOCTYPE html>

<html>

....

...

</html>

CSS:cascading style sheets

We can change the color of our texts by using style = “ color : ...”

For example <h2 style = “color: blue;”> CatPhotoApp </h2>

But there is a more global approach. We can define how to treat h2 tags with these codes.

<style>

h2{

color: red;

}

</style>

However, not being limited by only tags, we can also use classes and then while using tags, we can imply which class we are using. E.g; class definition start with .(period)

<style>

.red-text{

color : red;

}

</style>

Then while using <h2 class = “red-text”> .... </h2>

We can also change font size and family by using font-size = 16px and font-family = monospace (just a font).

If you are using a font more than one words you need to use “open sans” ( quotation marks) if not just type their name to font-family.

We can implement the font we want from google font by using a code like this one: it appears when you click “select this style” on the right;

<link href = “ the url of the font from google fonts” rel = “stylesheet” type = “text/css”>

We can apply multiple classes by one statement : class”class1 class2 ... “

You can round the borders, by using border-round: 10px ( or you can use percentages like 50%)

It is always a good practice to use unique id’s for some tags

For example: <h2 id = “new-header” class = “wider font” > .... </h2>

By using ids we can set properties accordingly in the style without using classes but by giving their ids by #

For example in style tag use #cat-photo-element{ background-color: green;}

Three important properties control the space that surrounds each HTML element: paddin, border and margin.

Margin adjusts space between element and surrounding elements while padding adjusts the space between the element border and inside elements.

Attribute selectors help us change the types default sizes or other properties. For example:

[type = “radio”] {

margin: 40px 20px 20px 40px;

}

By changing properties of <body> in the style, we can change every inherited things properties too.

İf we do color:green; inside body inside style, then all p’s and headers and etc will be green.

İd declerations have precedence comparing to class declerations. Inline styles overrides all of them. However if you use keyword !important in a class element, this overwrites all. For example

.pink-color{

Color: pink !important;

}

Creating variable: --penguin-skin : gray;

And when you are using you can apply it by background = var(--penguin-skin);

Text-align, textleri merkezde sağda solda ya da justify olarak her iki uca değecek şekilde ayarlamayı sağlar. E.x. text-aling:center; Justify, genel olarak kelime arası boşlukları gerekliyse artırıp her iki ucu da kenarlara değdirmeye çalışır.

* Örneğin başlığın (h) height ını artırırsan, onu yukarı çıkarmaz. Altındakileri ona göre aşağı indirir.

<strong> </strong> arasında kalan texti bold yapar.

<u> </u> underlines the text inbetween.

<em> </em> tag makes the text italic inbetween.

<s> </s> strikethrough. Kelimenin üzerini çizer.

<hr> creates a line. Used under headers usually.

rgba(0,0,0,0), used for R,G,B and A takes values between 0 and 1. A stands for opacity. It is good for using in background.

Opacity: 0.5 ( makes the opacity of what is selected half).

Text-transform: uppercase; makes the text uppercase;

Font-weight: is used for how thick or thin is the text going to be. Values are not px but numbers

For e.x.: font-size : 13px; font-weight: 200;

Line-height: 13px; arranges the spaces between the horizontal lines.

Hover: üzerine gelindiğinde olan şey. Pseudo class olarak oluşturulur. For ex.

  a {

    color: #000;

  }

  a:hover {

    color: #00F;

  }

Here we have a black href but when cursor is on it it becomes blue.

By using position: relative , we can use other locations in the page other than “natural flow” of what html offers. Positioning gives a lot flexibility. Usually used with left right top bottom keywords.

Position: absolute; this locks the element to its parent. Unlike relative, it is being removed from the natural flow so the fallowing elements ignore its presence.

Position:fixed; same with absolute, however, when users scrolls down or up, it also moves.

Float: left, float:right. It is used to make 2 or more things created on the same horizontal line by setting one on left and one to right.

For overlapping elements we use z index. Only takes integer values. Z-index: 2;

Margin:auto; makes the images and other stuff being centered horizontally.

Display: inline-block makes the two side by side again.

Hue: means color. Saturation: means amount of gray in a color. Full saturation no gray. Lightness: amount of white or black in a color. %50 is normal color.

Hsl stands for Hue, saturation, lightning. For example green code in hsl is hsl(120,100%,50%);

<nav> </nav> used to display multiple links.

nav ul {

    margin: 0px;

    padding: 5px 0px 5px 30px;

  }

  nav li {

    display: inline;

    margin-right: 20px;

  }

İf we are using it with list style, we can arrange ul and li like this.

Gradient color: first parameter is for direction, others are colors starts and ends. For ex.

Background: linear-gradient(35deg, red,yellow); There is also repeating-linear-gradient()

Using images from internet as background:

Between <style> body{ background: url(“ link”); } </style> ( background supports url function)

Transform: scale(2); makes the size of an element double.

Transform: skewX(x deg); skews the given element in x direction according to the degree

 box-shadow: 25px 10px 0 0 blue;

shadow properties, respectively, offest-x offset-y , blur-radius, spread-radius.

::before and ::after elements are usually used for doubleing an element. They are pseudoclasses and they must have a content: “”; usually an empty string is given.

margin: auto;

    top: 0;

    right: 0;

    bottom: 0;

    left: 0;

Use this to make an element centered at the middle of the page.

For animations two important things are animation-name: (a name you can give) and animation-duration: 4s (4 seconds animation) then you say @keyframes nameyougave{ and here you say 0%{} 50{} ...}

 @keyframes rainbow{

    0%{

      background-color: blue;

    }

     50%{

      background-color: green;

    }

    100%{

      background-color: yellow;

    }

  }

To make an element preserve its last condition at the end of an animation, we use animation-fill-mode:forwards;

I think setting only %50 keyframe makes the animation go and back automatically.

animation-iteration-count: infinite;

or just 3; this arranges how many times will the animation play.

animation-timing-function: linear;

animation takes the same time the is set in animation-duration buy with keywords linear, ease, ease-in, ease-out, the movement that it takes changes.

Also cubic-bezier() function is used for animation timing function. You pass 4 arguments that represents values of x and y cordinates of 2 points.

Semantic meaning: the tags we use indicates what information is stored in between them

Search engines look for only one <h1> tags. Therefore every web page should only contain 1 <h1> tag.

İf a book is <article> then every chapter is <section>.They can be used nested.

Audio implementation is usually used with “controls” and inclue <source src = “ horse.mp3” type= “ audio/mpeg>

While creating a form with radio elements, we should cover them with a <fieldset> and they usually contain <legend> as the informer and header and prompt of the given choices. Screen readers usually read this legend to understand what is in the poll.

For inputs there is another type other than text and submit, which is “date”.

<sup> </sup> usually used for dates. Makes for example th at the top right. 13 <sup> th </sup>

<time datetime = “2016-05-15”> Thursday, May 15 <sup>th</sup></time>

.sr-only{} in style is used for screen readers. İf you use visibility:hidden; then no one including screen readers can see. The .sr-only class hides element to all devices except screen readers.

<div tabindex = “0”>bla bla </div> here, when we are tabbing through texts, this makes this item tabable.

Media Queries is for different devices that runs the website. It is for visual optimization.

We are using @media (min-height: 800px) or @media(max-height: 800px), one implies if the device height minimum 800px then what you want to do. For example you can do p{10px;} with this, you can increase or decrease font size. You can also use @media (min,max-weight: ){}

Responsive images: when you zoom or enlarge the website, responsive images scale accordingly. For this here is a structured code: .responsive-image(can take another name, just a class){

Max-width: 100% (this guarantees that image will never be larger than its original size)

Height: autor (this guarantees that image size ratio will always be protected)

}

For high resolution devices, you should use half of the original height and width of the image. This is called retina image to make sure it will not be pixelized. For example if the image is 500x500, use 250x250 in your css.

Viewport units helps us arrange texts and images according to the devices. For that for exampel

İmg{

Width: 30vw; (vw means viewport width and here we sat it 30 percent of it, it should be minimum 10);

Height: 10vh (vh means viewport height and must be minimum 3percent so 3)

Width: 70vmin(viewport minimum, viewports smaller dimension, should be 70%)

Height: 100vmax(viewport maximum, viewports bigger dimension, should be 100%)

}

Display: flex; helps us flexibilty while arranging elements. Display:flex also has properties with it like flex-direction: row or column or row-reverse or column-reverse. By default it is row.

Sometimes this property doesnt fit the size of page, for this we can use for example justify-content:center; which alligns all items to center. Other than that flex-start, flex-end, space-between,space-around and space-evenly can be used.

Also there is align-items: property which can be set to lots of values like center;

Flex-wrap: nowrap, wrap, wrap-reverse;

When the container of elements has a width or length smaller than the sum of the elements that it contains, with flex-shrink property we can shrink the elements accordingly. İt takes integers and if one is with flex-shrink:1; the other is 2, then 2 will shrink 2 times more than the other.

When container expands we use flex-grow to grow the elements that it contains.

İf container has display:flex; the childs can have flex: flex-grow flex-shrink flex-basis respectively. For example flex: 2 2 150px; will set flex-grow to 2 and so on.

Order:1,2,3,..; sets the order of flex displayed items positions.

Another display feature is display:grid;

Afther giving grid, with grid-template-columns: 10px 10px ...; (if you give 10px 10px it will arrange the items inside the parent ( assuming you applied this to the container) arrange in order of 2 columns that are 50px wide.

Grid-template-rows: 10px 10px; arranges 2 rows that 10px height for the same items.

Fr(remained are fraction) auto(sets the column or row height or width according to its contenjt.

%(sets the column or row height or width according to the width and column of the containers that much percentage that holds the item)

İf you want gap between columns you can use grid-column-gap: 10px;

Similarly you can use grid-row-gap:10px;

Easier way: grid-gap: property, first one row gap, second one column gap. E.x: grid-gap: 10px 20px;

By using grid-column: 1/3; you can span an item from the first vertical line to the third one.

Similar with grid-row:

Justify-self: by default, while used in grid structure, this fills the area that are specified. But u can use start,center and end also. Justify-self:center; would center the property in the given space. This is fror horizontal arrangement.

Align-self:end; vertically alligns the item to the end in the given space.

Justify-items:center; arranges all items horizontally centered at their spaces.

Align-items: end; arrangs all items vertically end at their spaces.

By using grid-template-areas:

“ header header header”

“advert content content”

“footer footer footer”;

This creates template and it is written inside container. After doing this if we go to an item and then write grid-area: header; then that item will cover all header implied areas in the template.

Also grid-area: can be used like this 3/1/4/4 here respectively, horizontal start, vertical start, horizontal end, vertical end.

For example if we want to create grid-template-columns and grid-template-rows to create 100 areas, we use repeat function and its usage is repeat(num of times we need it to repeat 2, 1fr 50px) 20px;

This translates to 1fr 50px 1fr 50px 20px

There are auto-fit and auto-fill functions can be used with repeat. If necessary search for them