SASS

SASS Nedir?

SASS, Syntactically Awesome Stylesheets'in kısaltmasıdır. Bu, CSS (Cascading Style Sheets) dilinin bir önistemcisidir(preprocessor) ve CSS'e ek olarak daha avantajlı özellikler sunar.

Örneğin, SASS sayesinde CSS dosyalarınızı daha organize bir şekilde yazabilir, değişkenler kullanarak stil tanımlarınızı daha okunaklı hale getirebilir, fonksiyonlar oluşturarak tekrar eden stil tanımlamalarını önleyebilir ve daha fazlası.

SASS, CSS dosyalarınızı derleyerek standart CSS dosyalarına dönüştürür ve bunları web sitenizde kullanabilirsiniz.

Neden SASS Kullanmalıyız?

SASS, CSS dosyalarınızı daha organize ve okunaklı hale getirmeyi mümkün kılar. Bu sayede, daha büyük projelerde stil tanımlarınızı daha kolay takip edebilir ve değiştirme işlemini daha hızlı gerçekleştirebilirsiniz. SASS ayrıca, değişkenler kullanarak tekrar eden stil tanımlamalarını önleyebilir ve fonksiyonlar oluşturarak daha kapsamlı stil tanımlamaları yapabilirsiniz. Bu sayede, CSS dosyalarınız daha az kod içerecek ve daha anlaşılır hale gelecektir.

Özetle, SASS kullanarak CSS dosyalarınızı daha organize, okunaklı ve kapsamlı hale getirebilir, tekrar eden stil tanımlamalarını önleyebilir ve derleyerek standart CSS dosyalarına dönüştürebilirsiniz.

SASS’ ın Öne Çıkan Özellikleri

- Değişkenler
- İç İçe Öğeler(Nested Elements)
- Import (İçeri Aktarım)
- Mixin(Katman)
- Inheritance (birdençok dosya)
- Operatörler ve Hesaplamalar
- Fonksiyonlar

.scss ve .sass Farkı

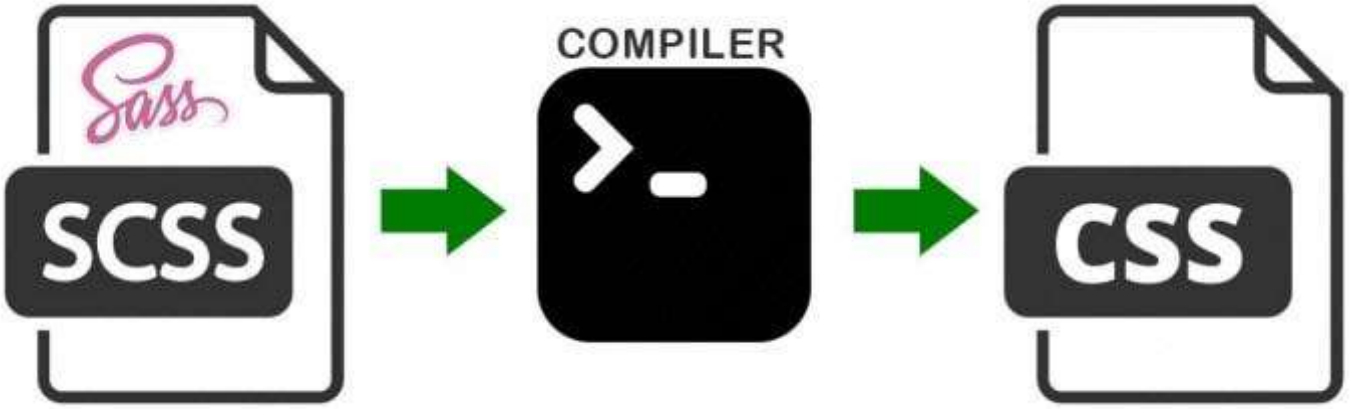
SCSS (Sassy CSS), SASS dilinin açık kaynak kodlu bir varyantıdır ve CSS sözdizimini takip eder. SCSS dosyaları, .scss uzantısı ile kaydedilir ve SCSS dilinde seçiciler, stil kuralları ve diğer öğeler parantez içine alınır, noktalı virgül kullanılır ve satır sonlarına (;) gerek duyulur. Bu nedenle, SCSS dosyaları CSS dosyaları gibi görünür ve daha anlaşılır hale gelir.

SASS ve SCSS arasındaki en önemli fark, SASS'ın özel sözdizimi ile SCSS'in CSS sözdizimini takip etmesidir. Bu nedenle, SASS dosyaları daha az kod içerebilir ve daha az okunaklı hale gelebilirken, SCSS dosyaları daha anlaşılır hale gelir. Hangisinin kullanılması gerektiği konusunda bir kesin kural yoktur ve tercih kişinin seçimine göre değişebilir. Ancak, SCSS'in CSS sözdizimini takip etmesi nedeniyle daha anlaşılır olması nedeniyle, genellikle SCSS tercih edilir.

SASS dosyalarında seçiciler ve stil kuralları parantez içine alınmaz , noktalı virgül ve çift tırnak kullanılmaz. Ayrıca, SASS dosyalarında derleme işlemi sırasında satır sonlarına(;) gerek duyulmaz.

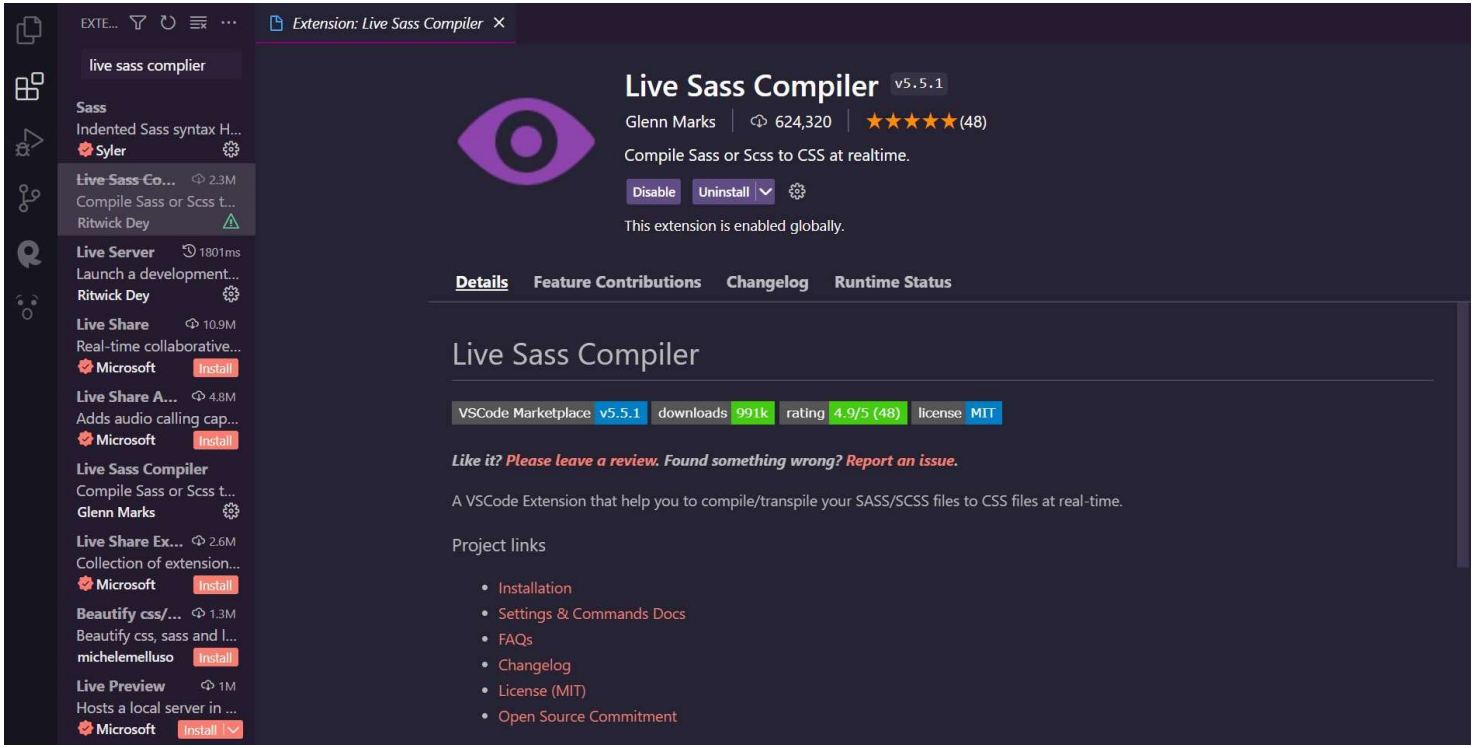
SASS Kullanımı

Tarayıcılar .scss / .sass uzantılı bir dosyayı direkt okuyamadıklarından dolayı bu dosyalara yazdığınız stilleri kullanmak için bir derleyici yardımıyla css dosyasına çevirmemiz gerek.



SASS Kurulumu

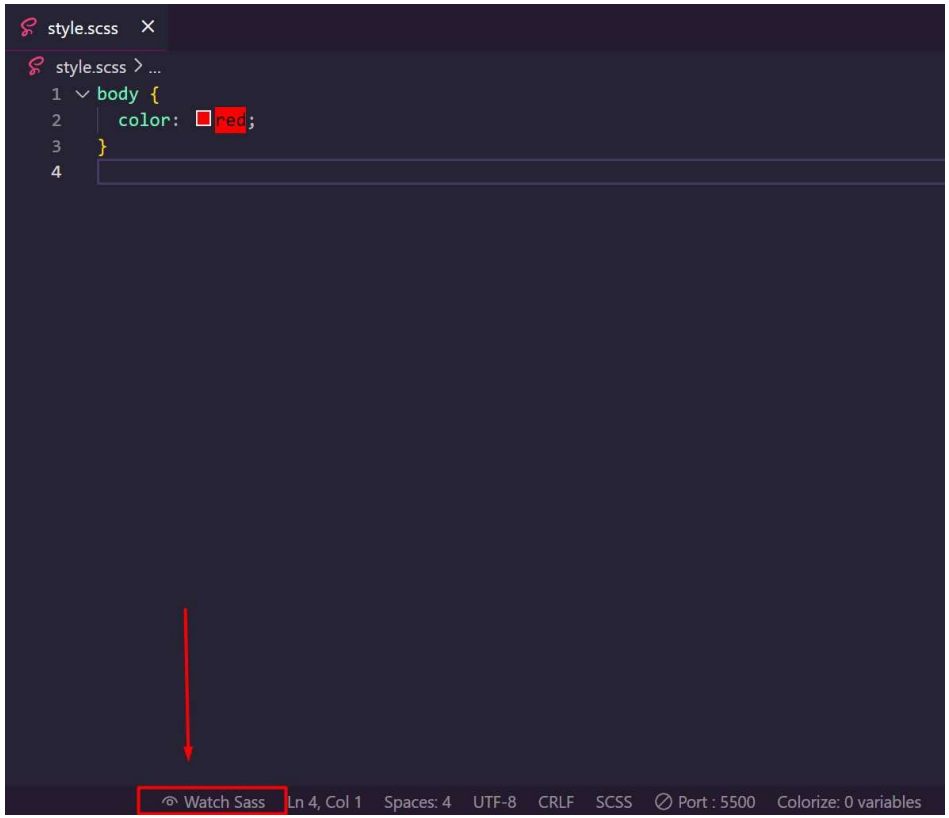
Öncelikle Visual Studio Code'a, Live Sass Complier eklentisini kurun.



Projenize .scss uzantılı bir dosya oluşturun

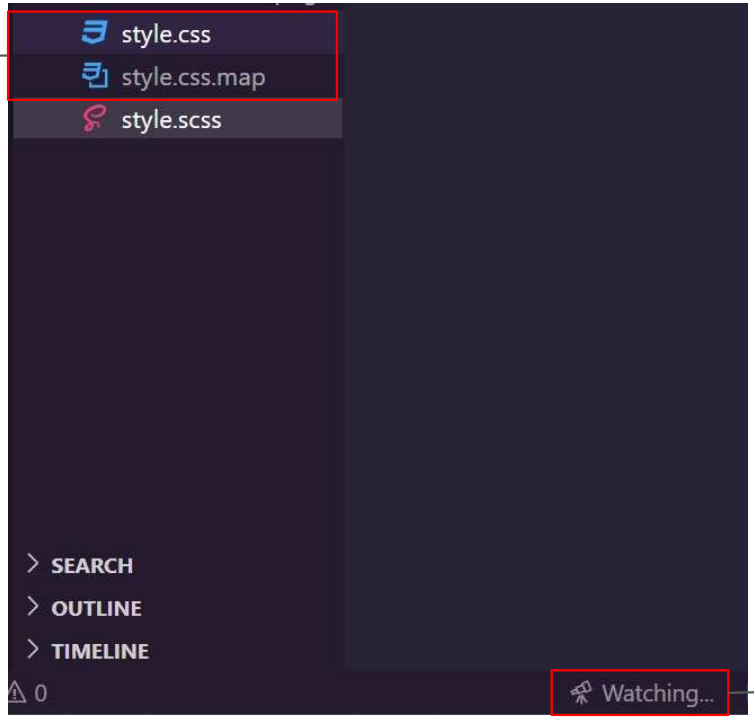
Oluşturduğunuz dosyaya geçici olarak bir css kodu yazın

SCSS'i , daha önce indirdiğimiz eklentiyle birlikte gelen, editörün alt kısmındaki bu butona basarak css dosyasına derleyelim



Compiler, SCSS dosyanızı CSS e derleyicek ve size iki tane css dosyası oluşturucak.Ve SCSS dosyasını izlemeye devam edip yapılan değişekleri css e derlemeye devam edicek.

oluşturduğu css dosyaları



izlediğini gösteren yazı

Artık scss dosyasında istediğiniz değişiklikleri yapabilirsiniz .

Editörünüzü kapatana kadar complier(derleyici) görevini yapıp scss te yaptığınız değişiklikleri css için derleyecek.

Editörü tekrar açtığınızda ise watch sass butonuna basarak tekrar devreye sokabilirsiniz derleyiciyi.

Variables(Değişkenler)

SASS sayesinde, stil tanımlarınız için değişkenler oluşturabilirsiniz.
Değişkenler, stil tanımlarınızda tekrar eden değerleri (örneğin renkler, fontlar, ölçüler vb.) tutmak için kullanılan etiketlerdir.
Bu sayede, stil tanımlarınızda tekrar etmek zorunda kalmayacak ve değişiklik yapmak daha kolay hale gelecektir.

Değişkenler, aşağıdaki gibi tanımlanır:

```
$degisken-adi: değer;
```

Örneğin, bir renk değişkeni oluşturmak için aşağıdaki gibi bir tanım kullanılabilir:

```
$ana-renk: #f00;
```

Bu değişkeni kullanmak için, stil tanımı içinde değişken adının önüne "$" işareti koyulur. Örneğin:

```
.baslik {
  color: $ana-renk;
}
```

Bu şekilde, .başlık sınıfının rengi olarak $ana-renk değişkeninin değeri kullanılır.

Eğer $ana-renk değişkeninin değeri değiştirilirse, .başlık sınıfının rengi de otomatik olarak değişecektir.

Değişkenler, SASS sayesinde stil tanımlarınızı daha okunaklı hale getirmeyi ve değişiklik yapmayı kolaylaştırmayı sağlar.

Nested (İç İçe) Elements

SASS sayesinde, stil tanımlarınızı daha okunaklı hale getirebilir ve işlemenizi kolaylaştırabilirsiniz.

Bunu yapmak için, iç içe öğeler adı verilen yapıyı kullanabilirsiniz.

İç içe öğeler, bir stil tanımı içinde birden fazla seçici kullanmak istediğinizde, tek bir satırda yazmanızı sağlar.

Bu sayede, stil tanımlarınız daha okunaklı hale gelir ve tekrar eden kodları önlersiniz.

Şöyle bir html kodumuz olduğunu varsayalım

```
<div class="ana-sekme">
  Bu alan kapsayıcı elemente aittir
   <div class="alt-sekme">
     Çocuk Element
   </div>
</div>
```

SCSS te kapsayıcı elementin içindeki diğer elementleri aynı html de yaptığımız gibi iç içe yazarız.

```scss
.ana-sekme {
  color: black;

  .alt-sekme {
    color: gray;
  }
}
```

Aynı yapının css te kullanımı şu şekildeydi

```css
.ana-sekme {
  color: black;
}

.alt-sekme {
    color: gray;
}
```

Daha iyi anlamanız için somut bir örnek oluşturalım.
Aşağıdaki gibi bir html yapımız olduğunu düşünelim

```
<ul>
   <li><a href="">Anasayfa</a></li>
   <li><a href="">Hakkımızda</a></li>
   <li><a href="">İletişim</a></li>

  </ul>
```

Bu durumda saf CSS kullanarak üç elemana stil özellikleri vermek için aşağıdaki gibi bir yapıya ihtiyacımız vardır.

```
ul {
    margin: 0;
    padding: 0;
}
ul li {
    float: left;
    list-style: none
}

ul li a {
    background: #222;
    color: #fff}
```

Görüldüğü gibi iç içe yapılar için seçiciler yazılırken tekrarlar meydana gelmektedir. İşte SASS ve SCSS bu durumda oldukça pratik bir kullanım sunmaktadır.

Yukarıdaki kod bloğunun SASS'taki karşılığı aşağıdaki gibidir.

```
ul
    margin: 0
    padding: 0

    li
        float: left
        list-style: none

        a
            background: #222
            color: #fff
```

CSS'de ul li veya ul li a şeklinde uzayan seçiciler SASS'ta görülmemektedir. Bu örnekte her bir girinti o elemanın bir üst elemana ait olduğunu belirtir. Bu yüzden yazarken dikkatli olmamız gerekmektedir.

Gelelim aynı örneği SCSS'de yapmaya

```
ul {
margin: 0;
padding: 0;

 li {
  float: left;
  list-style: none;

    a {
     background: #222;
     color: #fff;}
  }
}
```

SCSS'de süslü parantezler scope yani kapsama alanını belirlediği için girintilerin bir önemi yoktur. Ancak bir eleman, süslü parantezleri içinde bulunduğu elemana bağlıdır. Bunu unutmamanız gerekmektedir. Yine de düzgün görünmesi için girintilere önem verebilirsiniz.

SCSS'de de SASS'ta olduğu gibi seçici isimleri ul li veya ul li a gibi uzamadı. Sadece iç içe geçmiş üç elemana stil vermek için bile bu kadar kolaylık sağladığını göz önünde bulundurarak iç içe geçmiş 4-5 elemanda ne kadar büyük kolaylık sağlayacağını tahmin edebilirsiniz.

Import (İçeri Aktarım)

Kod kalabalığından ve karışıklıktan kurtulmak için çok kullanışlı bir özelliktir. 2 adet .scss uzantılı dosyamız olsun.

Birinde genel tanımladığımız değişkenlerimiz bulunsun, bir diğerinde ise still verdiğimiz SASS kodlarımız bulunsun.

Bu özellik ile değişkenleri tanımladığımız _variable.scss dosyasını, still verdiğimiz style.scss dosyası ile birleştirmiş oluyoruz.

```scss
//main.scss

/* base */
@import "base/reset";
@import "base/typography";

/* components */
@import "components/buttons";
@import "components/nav";
@import "components/dropdown";
```

_variables.scss

```scss
$font-rengi: yellow;
$font-boyutu: 32px;
$font-kalınlığı: bold;
```

style.scss

```scss
@import "variables";

h1{
 font-size: $font-boyutu;
 font-weight: $font-kalınlığı;
 color: $font-rengi;
}
```

Yukarıdaki kodların CSS e derlenmiş hali

```
$font-rengi: yellow;
$font-boyutu: 32px;
$font-kalınlığı: bold;

h1{
 font-size: $font-boyutu;
 font-weight: $font-kalınlığı;
 color: $font-rengi;
}
```

Mixin(Katmanlar)

Katman yapısı, sürekli aynı parametreleri kullanarak yazdığımız CSS kodlarını tek bir sefer tanımlayarak, her yerde tek bir komut çağırarak kullanmamıza olanak sağlar.

Fonksiyon çağırmaya çok benzer fakat burada işlem yaptırmıyoruz.

Basit bir örnek ile başlayalım;

```
@mixin boyut {
   width: 100px;
   height: 200px;
}
```

Mixin ile tanımladığımız stilleri çağırmak için @include komutu kullanılır

```
.ürün{
 background-color: red
 @include boyut
}
```

Yukarıdaki kodların CSS e derlenmiş hali;

```
.ürün{
 background-color: red
 width: 100px
 height: 200px
}
```

Farklı bir örnekle devam edelim. Bu sefer aynı mixin yapısına parametre gönderelim. Mixin yapılarında değişkenler sayesinde parametre alabilmekteyiz.

```
@mixin boyut($genislik, $yukseklik) {
    width: $genislik;
    height: $yukseklik;
}
.ürün {
    background-color: red
    @include boyut(200px,400px) }
```

Son örneğimizde mixin in içinde @content kullanarak ekran 760px pencere boyutuna ulaşana kadar vereceğimiz stilleri istediğimiz elementte çağırarak yazıcaz.

```
@mixin mobile {
  @media screen and (max-width: 760px) {
    @content;
  }
}
```

@include ile @content sayesinde istediğimiz değerleri giriyoruz

```
.container {
  min-width: 400px;
  width: 65%;
  height: 75%;
  @include mobile {
    height: 90%;
    min-width: 310px;
  }
}
```

Daha önce mixin kullanarak belirlediğmiz ekran boyutuna kadar geçerli olucak stilleri tanımladık

Inheritance(Miras Alma)

Katılım özelliği ortak CSS kodlarını barındıran elementleri bir araya toplamak için kullanılanılır.

Örneğin daha önce bir class için stil tanımladınız daha sonra farklı bir elementi stilize ederken önceden tanımladığınız class ın stillerini alıp üzerine farklı stiller eklemek istediniz. Bu durumda @extend devreye giriyor.

Örneğin bir mesaj kutucuğu oluşturmak istediniz

```
.mesaj{
 border: 1px solid gray;
 padding: 15px;
 color: black;
}
```

Ve ardından yine aynı stilleri alan farklı varyantlarını oluşturdunuz;

```
.basarili-mesaj{
 @extend .mesaj;
 background: green;
}
.hata-mesaj{
 @extend .mesaj;
 background: red;
}
```

Operators(Operatörler)

Web sayfasına bölümleri yerleştirirken çeşitli hesaplamalar yapmamız gerekebiliyor.

SASS ile hesaplamalar için +, -, *, / ve % operatörlerini kullanabiliriz.

SASS operatörlerini kullanarak sayısal değerler üzerinde işlem yapabileceğimiz gibi renk değerleri içinde işlem yapmamıza imkan vermektedir.

Bu işlemlere ek olarak SASS içinde yuvarlama fonksiyonlarında çalışmaktadır. Bu fonksiyonlara örnek olarak percentage(), floor() ve round() fonksiyonu örnek verilebilir.

Örneğin; sayfamızda 3 tane kutu modeli olsun.

```
<div class="box-1">box-1</div>
<div class="box-2">box-2</div>
<div class="box-3">box-3</div>
```

Kutuların ortak özelliklerini bir sınıfta tanımlayıp SASS @extend ile miras alarak diğer sınıflara uygulayalım.

```
$box-padding: 50px;
$box-color: crimson;

.box {
  display: inline-block;
  color: white;
}
```

```scss
.box-1 {
  @extend .box;
  padding: $box-padding;
  background-color: $box-color;
}

.box-2 {
  @extend .box;
  padding: $box-padding+30;
  background-color: $box-color*2;
}

.box-3 {
  @extend .box;
  padding: $box-padding*2;
  background-color: $box-color/2; }
```

Oluşturduğumuz box-1/2/3 te daha önce tanımladığımız box classının stillerini extend ile alıp ardından istediğimiz şekilde özelleştirdik.

Ek olarak renkler üzerinde operatör kullanırken rengi bölmek rengi daha koyu, rengi çarpmak daha açık bir renk ortaya çıkaracaktır.

AND (&) Kullanımı

& işareti ile stillendirdiğiniz elementin seçicisini çağırabilirsiniz.

Bu sayede aynı kelimeler ile başlayan seçici(class, id) isimlerine müdahale ederken veya hover, focus gibi özellikler kullanırken sizi hızlandırır.

Şöyle bir html kodumuz olduğunu düşünelim

```
<div class="card">
  <div class="card-head"></div>
  <div class="card-body"></div>
</div>
```

Bu durumda bütün class ismini yazmak yerine class isimlerinin başlangıcı (card) aynı olmasından dolayı şu şekilde müdahale edebiliriz

```scss
.card {
  width: 400px;
  height: 400px;
  &-head {
    background: black;
  }
  &-body {
    background: white;
  }
}
```

Farklı bir örnekte bir buton tanımladığınızı varsayalım

```
<button class="paylas">Paylaş</button>
```

Bu butona hover efekti vermek istersek;

```
.paylas{
  padding: 20px;
  &:hover{
    background-color: gray;
  }
}
```

Functions

SASS dilinde, fonksiyonlar kullanılarak, tekrar eden işlemleri ve kodları önleyebilirsiniz. Fonksiyonlar, belirli bir işlemi gerçekleştirir ve sonucunu döndürür. Bu sayede, fonksiyonları tekrar tekrar kullanarak kod tekrarını önleyebilir ve stil tanımlarınızı daha okunaklı hale getirebilirsiniz.

SASS fonksiyonları, aşağıdaki gibi tanımlanır:

```
@function isim($parametre1, $parametre2, ...) {
 // Fonksiyon işlemleri
 @return sonuc; }
```

Örneğin, bir font boyutu fonksiyonu oluşturmak için aşağıdaki gibi bir tanım kullanılabilir:

```
@function font-size($renk) {
  @if $renk == red {
    @return 18px;
  } @else {
    @return 14px;
  }
}
```

Bu fonksiyon, bir parametre olarak renk alır ve eğer renk "red" ise, font boyutunu "18px" döndürür. Eğer renk "red" değilse, font boyutunu "14px" döndürür. Bu fonksiyonu kullanmak için aşağıdaki gibi bir stil tanımı yapılabilir:

```
.baslik {
  color: red;
  font-size: font-size(red);
}
```

Bu şekilde, .başlık sınıfının font boyutu "18px" olur. Eğer renk değiştirilirse, font boyutu da otomatik olarak değişecektir.

SASS fonksiyonları sayesinde, stil tanımlarınızı daha okunaklı hale getirebilir ve tekrar eden kodları önleyebilirsiniz. Ayrıca, fonksiyonları tekrar tekrar kullanarak kod tekrarını önleyebilir ve stil tanımlarınızı daha anlaşılır hale getirebilirsiniz.

Daha iyi nlaşılması için farklı birkaç fonksiyon daha yazalım.

***Renk çevirme fonksiyonu*:**

```
@function to-rgba($renk, $opaklik) {
  $rgb: color($renk);
  $alpha: $opaklik / 100;
  @return rgba($rgb, $alpha);
}
```

Bu fonksiyon, bir renk ve opaklık değeri alır ve bu değerleri kullanarak RGBA formatta bir renk döndürür. Örneğin:

```
.baslik {
  color: to-rgba(red, 50);
}
```

Bu örnekte, .başlık sınıfının rengi opak bir kırmızı olacaktır.

Ölçü birimi çevirme fonksiyonu:

```
@function to-rem($deger, $anakazanci:
16px) {
  @return $deger / $anakazanci * 1rem;
}
```

Bu fonksiyon, bir ölçü birimini "rem" birimine çevirir. Örneğin:

```
.baslik {
  font-size: to-rem(24px);
}
```

Bu örnekte, .başlık sınıfının font boyutu "1.5rem" olacaktır.

Ölçü birimi karşılaştırma fonksiyonu:

```
@function greater-than($deger1,
$deger2) {
  @if $deger1 > $deger2 {
    @return true;
  } @else {
    @return false;
  }
}
```

Bu fonksiyon, iki ölçü birimini karşılaştırır ve eğer ilki büyükse "true", değilse "false" döndürür. Örneğin:

```
@if greater-than(100px, 50px) {
  .baslik {
    font-size: 24px;
  }
} @else {
  .baslik {
    font-size: 18px;
  }
}
```

Bu örnekte, .başlık sınıfının font boyutu "24px" olacaktır.

Bu örnekler, SASS dilinde nasıl fonksiyonlar oluşturulabileceğini ve bu fonksiyonları nasıl kullanabileceğinizi gösterir.

SASS fonksiyonları sayesinde, stil tanımlarınızı daha okunaklı hale getirebilir ve tekrar eden kodları önleyebilirsiniz.

Ayrıca, fonksiyonları tekrar tekrar kullanarak kod tekrarını önleyebilir ve stil tanımlarınızı daha anlaşılır hale getirebilirsiniz.