```
<!DOCTYPE html>
<html lang="pt-br">
<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">
<link rel="stylesheet" href="sass/css/debug.css" />
<title>Regra @at-root</title>
</head>
<body>

<!--

            Regras de Estilo @at-root

        O que faz a Regra de Estilo @at-root?

    A regra de estilo @at-root (do inglês: “Na Raiz”) nos permite fazer com que tudo o que seja escrito dentro dessa
    regra seja criado no contexto da raiz, mesmo que esteja dentro de um alinhamento. Essa regra nos permite criar
    seletores específicos sem gerar grandes especificidades para isso, nos permite economizar linha de código e gerar
    um código mais profissional.
    Sem falar que facilita a nossa vida de queremos gerar um estilo específico num código que já está implantado,
    imagine o trabalho que seria desmembrar o código em mais linhas gerando novos alinhamentos, em vez disso, usando a
    regra @at-root, nós simplesmente podemos criar um mixin que é capaz de gerar formatações na raiz do documento usando
    a regra @at-root dentro do mixin.


        Como utilizar a Regra @at-root?

    Podemos utilizá-la de duas maneiras, de maneira Local no bloco de desejamos gerar uma formatação de estilo raiz, ou
    por referência, gerando um mixin que contenha a formatação necessária para gerar um seletor raiz dentro de um bloco
    já existente.
    Vejamos as 2 formas de utilização de @at-root logo abaixo...

    * Gerando @at-root no contexto local...
```

Arquivo HTML

```
<body>

<form class= “wrapper”>
<input class=“field” type=“text”/>
<input class=“field” type=“email”/>
<selected class=“field”>
        <option>Brasil</option>
</selected>
<input type=“checbox”/>
</form>

</body>
```

Arquivo Sass: style.scss

```
@use “sass:selector”;

.wrapper {
    display: flex;
    flex-direction: column;
    .field: {
        @at-root #{selector.unify(&, “input”)} {
            padding: 10px;
            border: 1px solid #999;
            margin: 10px;
        }
    }
}
```

Perceba no exemplo acima que a primeira coisa que fizemos na “linha 1” foi chamar para biblioteca nativa do Sass “selector”. Por que chamamos essa biblioteca? Por que ela contém uma função necessária para fazer a união do seletor do contexto atual – que no caso é o seletor “.field” - com o seletor do contexto raiz do documento que no caso é o “wrapper”.

O que desejamos fazer aqui é fazer com que cada elemento HTML que passarmos dentro da função “unify” crie um seletor dentro do contexto raiz caso ele tenha a classe “field” declarada sobre ele.

No caso de cima, estamos gerando uma formatação raiz sobre o elemento “input”, e estamos usando o “&” para referenciar ao contexto local do bloco, que no caso é a classe “.field”. Em palavras mais simples estamos dizendo: “se qualquer elemento “input” dentro do meu “wrapper” tiver a classe “.field” declarada sobre ele, aplique sobre ele a formatação de estilo abaixo..”

> OBS: Perceba que após o @at-root estamos usando um template #{} (hashtag chaves), esse template serve para mostrar ao Sass que uma expressão Sass deve ser invocada, estamos usando ela aqui, por que estamos usando uma expressão Sass dentro de uma regra que não possuí expressão antes dos brackets dela, por isso existe a necessidade de sempre utilizar o template entre a regra @at-root e os seus brackets.

O resultado disso código final será...

```
.wrapper {
display: flex;
flex-direction: column;
}

.wrapper input.field {
padding: 10px;
border: 1px solid #999;
margin: 10px;
}
```

Perceba que as formatações foram criadas, e invés da formatação de input ficar “.field input” – como se a tag input fosse filha de um elemento que tivesse a classe “.field” – a formatação ficou “input.field”, mostrando que é o elemento “input” que tem a classe “field”.

Porém, esse método de implementação da regra no escopo local têm uma desvantagem, nós teríamos que repetir esse processo para todos os elementos dentro do “wrapper” que tiverem a classe “.field” caso desejássemos que uma formatação fosse aplicada. Essa desvantagem já é pode ser sanada quando utilizamos @mixin junto com @at-root, como podemos ver no segundo modo de utilização logo abaixo...

* Gerando @at-root no dentro de um mixin...

Arquivo Sass: style.scss

```
@use “sass:selector”;

    @mixin field($child) {

    @at-root #{selector.unify(&, $child)} {
    padding: 10px;
    border: 1px solid #999;
    margin: 10px;

}

.wrapper {
    display: flex;
    flex-direction: column;
    .field: {
        @include field(“input”)
        @include field(“select”)
        }
    }
}
```

Perceba que usando um mixin e a regra @at-root dentro dele, nós podemos simplesmente referenciar o mixin dentro do contexto do bloco onde desejamos gerar uma formatação de estilo global, passando dentro do @include o elemento que desejamos gerar a formatação raiz, em vez de criar uma formatação de estilo @at-root para cada um dos elementos dentro do bloco local.

Perceba a grandiosidade disso, mesmo que tenhamos outros elementos dentro do “wrapper” que contém o mesmo nome de “classe”, se eles não tiverem sido declarados por através do mixin, eles não terão a mesma formatação que áqueles elementos HTML que foram declarados dentro do mixin.

* Usando @content junto com @at-root no dentro de um mixin...

```
Poderá haver casos onde não desejaremos fazer a declaração das propriedades CSS dentro da regra @at-root que
estiver no mixin, vamos preferir fazer com que o mixin puxe as formatações que estiverem dentro do contexto do
bloco. Para fazer isso acontecer podemos usar a regra @content, dessa forma...

    Arquivo Sass: style.scss

    @use “sass:selector”;

    @mixin field($child) {

        @at-root #{selector.unify(&, $child)} {
            @content;
    }

    .wrapper {
        display: flex;
        flex-direction: column;
        .field: {
            @include field(“input”)
            @include field(“select”)
            padding: 10px;
            border: 1px solid #999;
            margin: 10px;
            }
        }
    }

Perceba que no mixin acima, não temos mais as propriedades CSS, em vez disso temos apenas a regra @content, essa
regra diz ao mixin que ele deve herdar a formatação que estiver em “&”, que é o contexto local do lugar onde o
mixin for referenciado – que no caso será “.field”.

OBS: Para mais detalhes veja "Fundamentos do Sass nº19";

-->
```

```
    <form class= “wrapper”>
        <input class=“field” type=“text” placeholder="Nome" />
        <input class=“field” type=“email” placeholder="Email" />
        <select class=“field”>
                <option>Brasil</option>
        </select>
        <input type="checkbox"/>
    </form>

</body>
</html>
```

ARQUIVO SASS...

```
@use "sass:selector"; /*Sempre devemos usar a biblioteca nativa "selector", ela que vai nos ajudar unificar os
                        seletores passados no mixin com o seletor raiz*/

@mixin field($child) {
    @at-root #{selector.unify(&, $child)} { /*Perceba o uso da template entre a regra @at-root e os brackets dela,
                                              além disso, temos a função unify da biblioteca "seletor" que unicará
                                              o seletor de contexto global com o elemento passando na variável
                                              $child*/
        padding: 10px;
        border: 2px solid #999;
        margin: 10px;
        @content; //O content vai criar outro seletor raiz com a formatação que estiver dentro do seletor filho.
    }
}

.wrapper {
    display: flex;
    flex-direction: column;
    .field {
        @include field("input"); //Inclusão do elemento "input"
        @include field("select"); //Inclusão do elemento "select"
```

```
        color: red;
    }
}
```

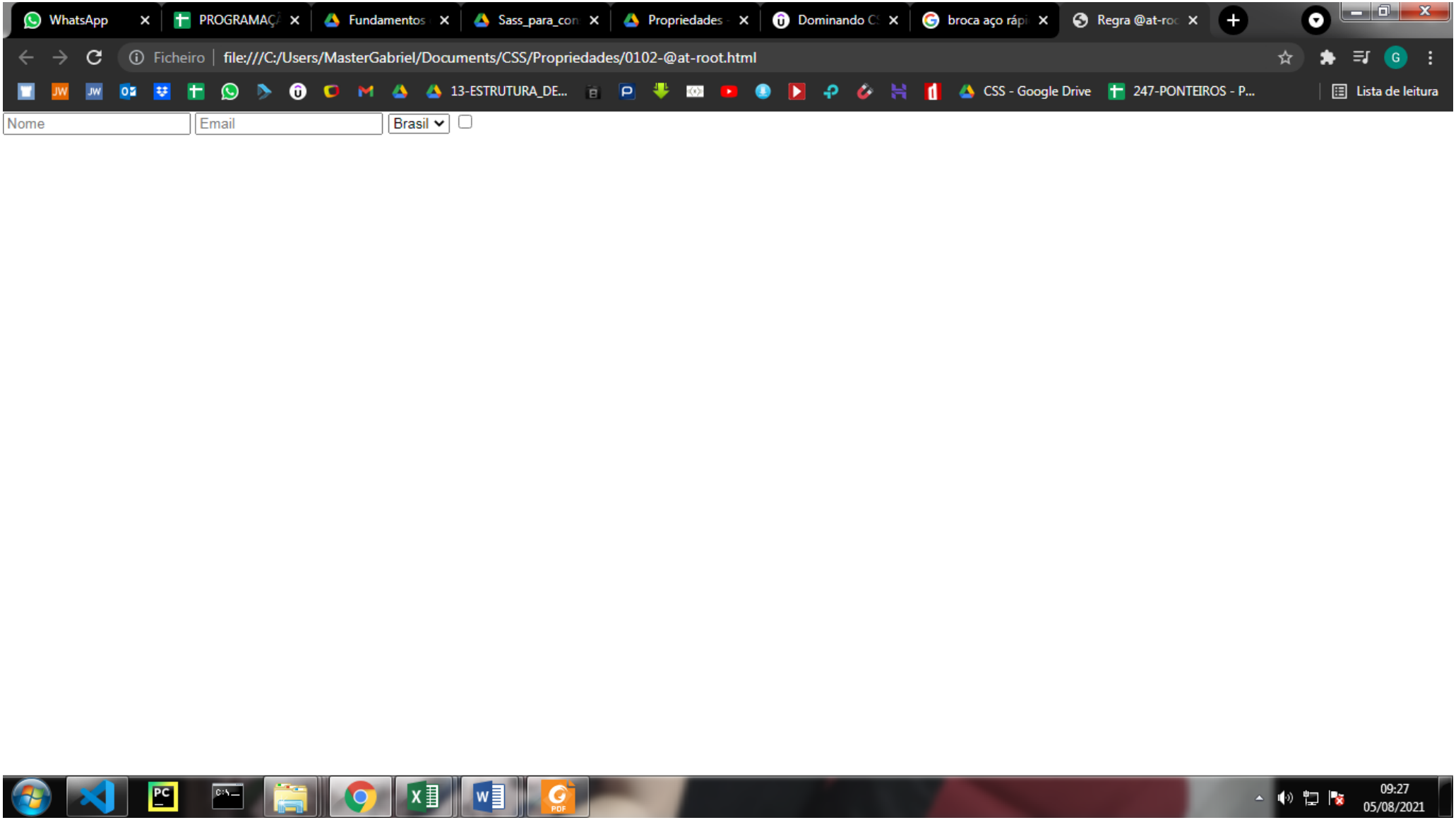
ARQUIVO CSS FINAL...

```
@charset "UTF-8";
.wrapper {
  display: flex;
  flex-direction: column;
}
.wrapper .field { /*Seletor criado graças ao @content*/
  color: red;
}
.wrapper input.field { /*Perceba que o elemento input foi criado como raiz tendo a classe "field" em seu elemento*/
  padding: 10px;
  border: 2px solid #999;
  margin: 10px;
}

.wrapper select.field { /*Perceba que o elemento select foi criado como raiz tendo a classe "field" em seu elemento*/
  padding: 10px;
  border: 2px solid #999;
  margin: 10px;
}

/*# sourceMappingURL=at-root.css.map */
```

RESULTADO NO NAVEGADOR LOGO ABAIXO...


WhatsApp
PROGRAMAÇ
Fundamentos
Sass_para_con
Propriedades
Dominando C
broca aço rápi
Regra @at-roc
Ficheiro | file:///C:/Users/MasterGabriel/Documents/CSS/Propriedades/0102-@at-root.html
13-ESTRUTURA_DE...
CSS - Google Drive
247-PONTEIROS - P...
Lista de leitura
Nome
Email
Brasil
09:27
05/08/2021