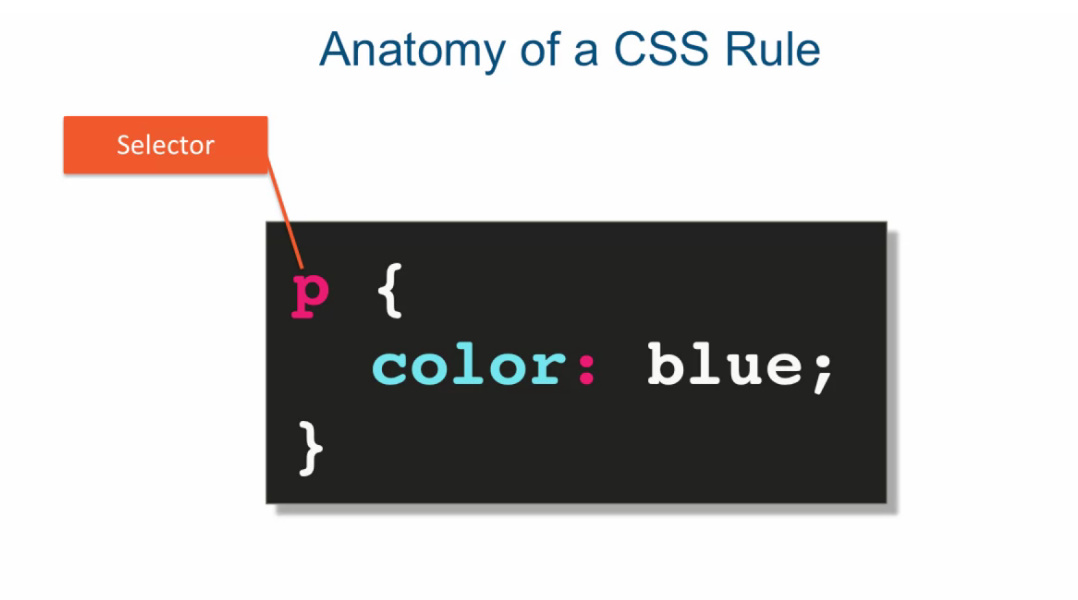
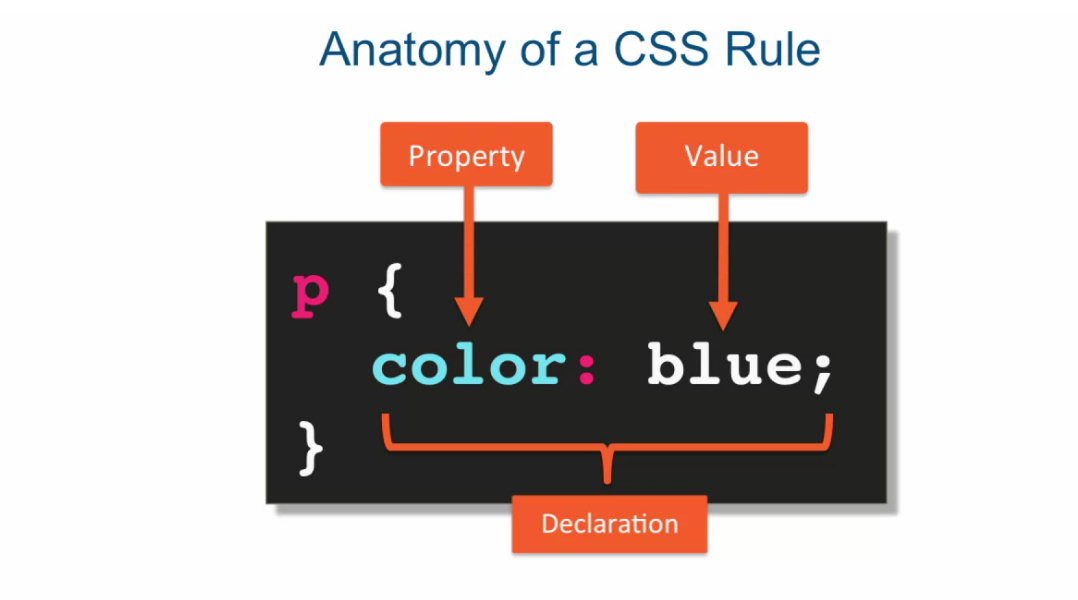
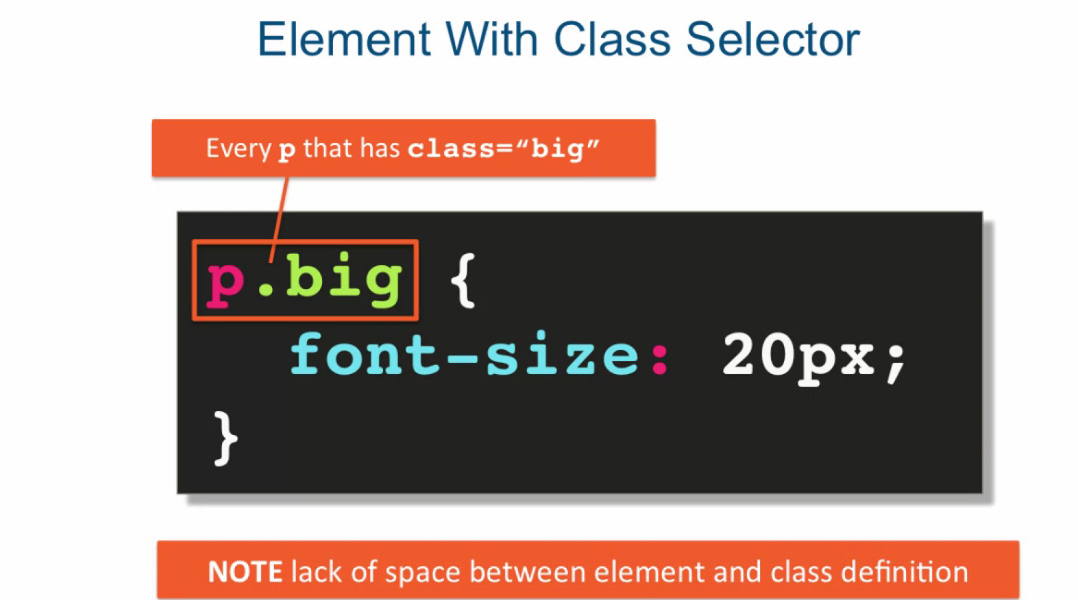
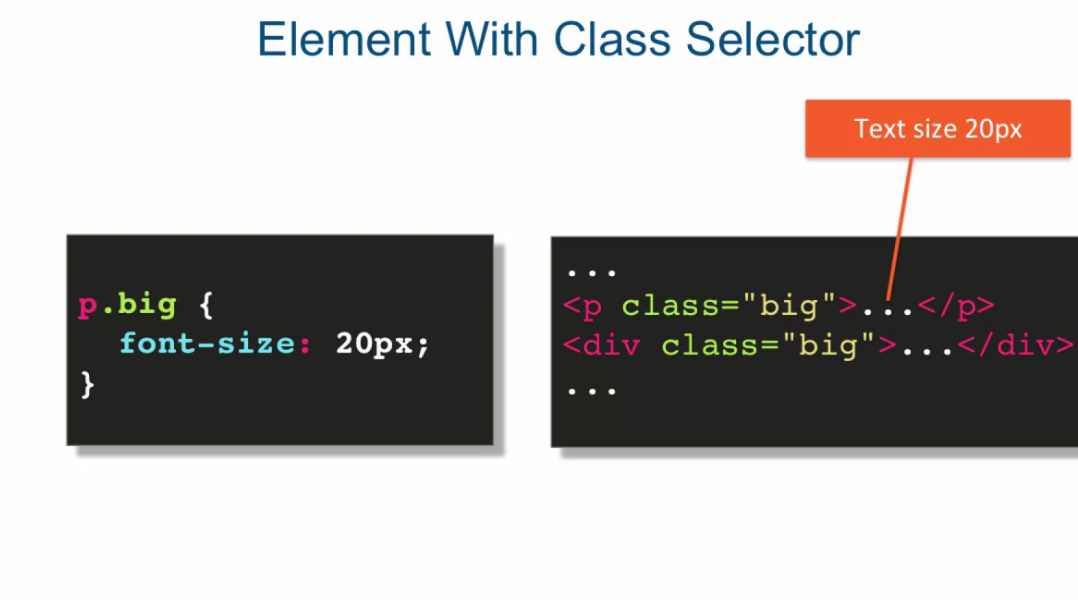
**CSS Anatomy**



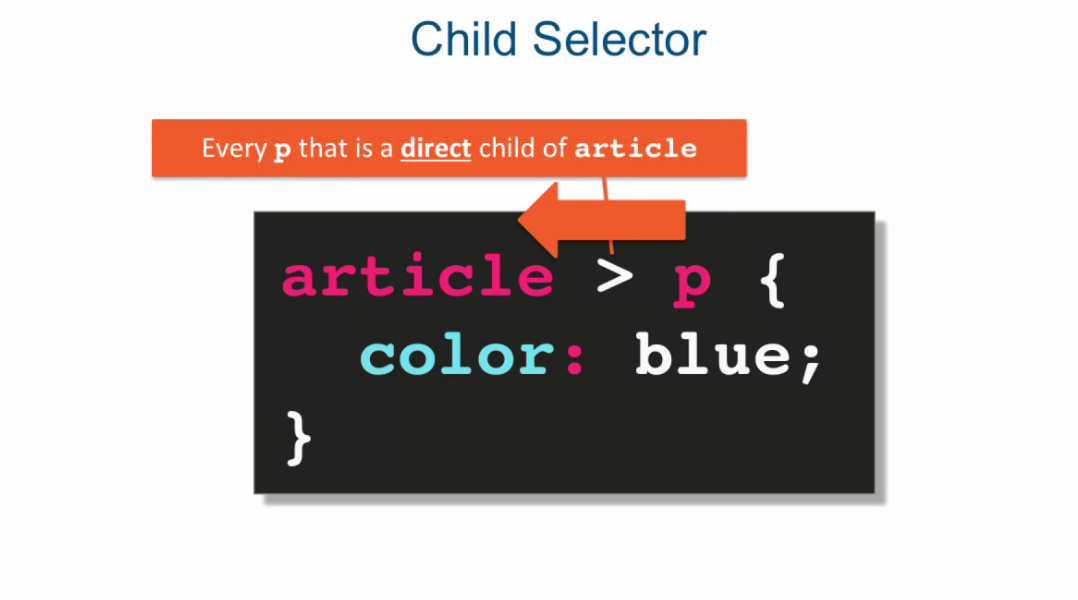


**Combining Selectors**

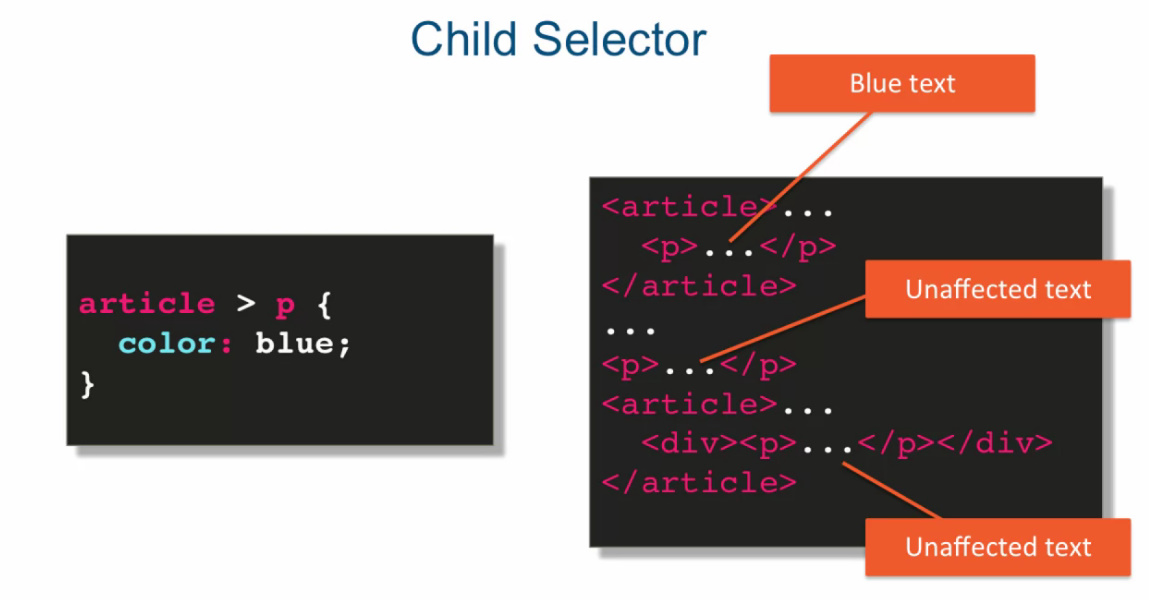




* **Child selector**



Este pentru fiecare P element care este un direct child a lui article

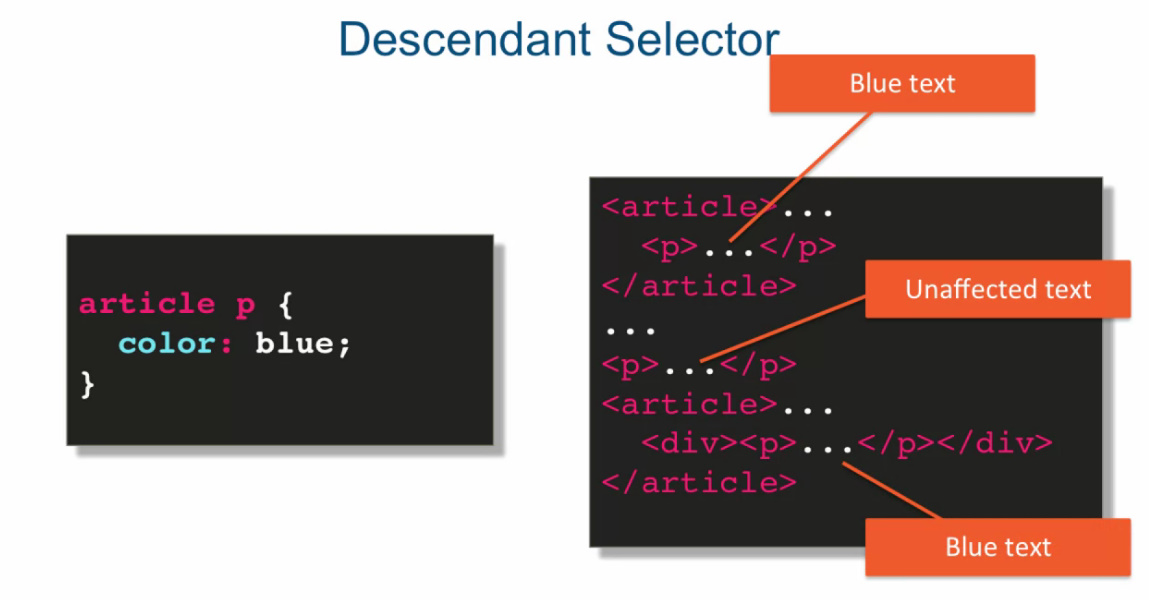


Deci, p trebuie sa fie un copil doar a lui article, nu si a altor elemente, fie ele si nested chiar.

* **Descending Selector**



Regula data se aplica oricarui element p din article, indiferent ca e copil direct sau nu





* multiple classes

.clasa1.clasa2{

}

Deci, se aplica elementelor ce au clasa1 si clasa2

De ex: <div class=”clasa1 clasa2”>

* first child

#menu-categories-title + div {

  margin-bottom: 50px;

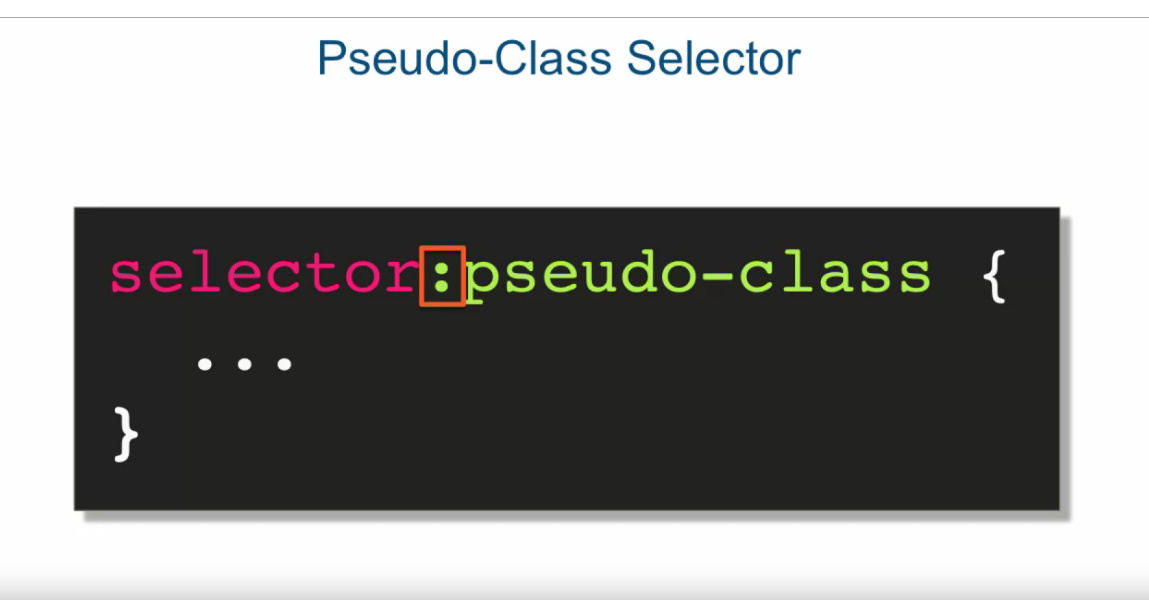
}

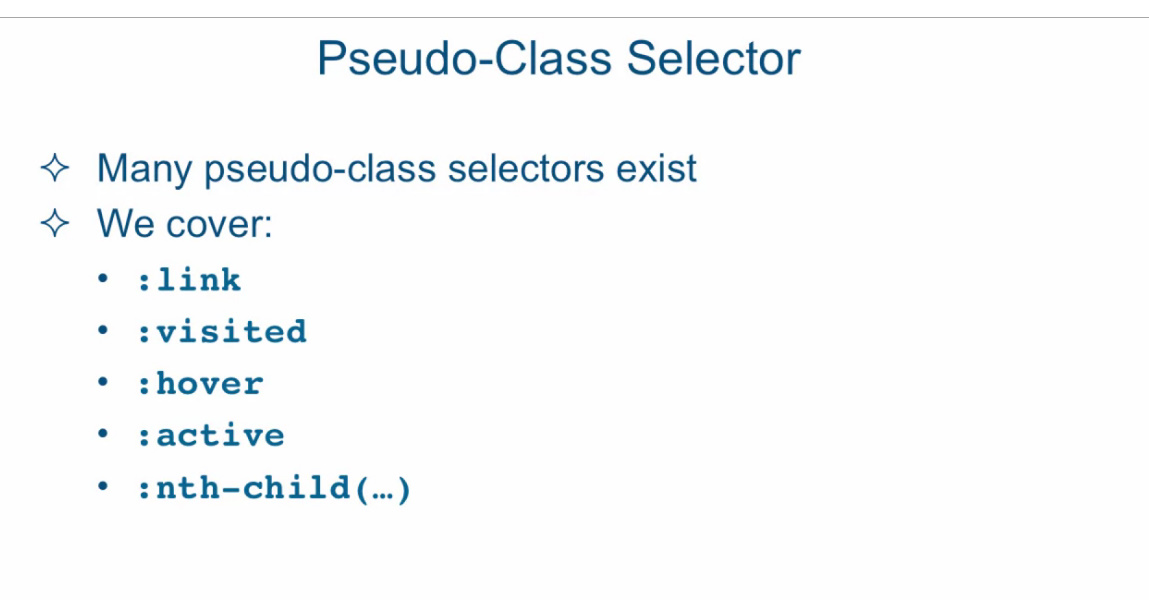
stilurile se aplica la primul copil <div> din elementul cu id menu-categories-title

**Multiple classes**

* <div class=”clasa1 clasa2 clasa3....”>

**Pseudo Class**





* header:nth-child(2){

}

Deci se aplica doar pentru copilul al 2 din header. Restul nu sunt afectate

* header:nth-child(odd){

}

Se aplica doar copiilor impari

* even e par
* Putem combina pseudo clasele:

div:nth-child(5):hover { ... }

**Conflicts**

* Cand definim mai multe CSS styles, mereu ultimul e cel luar. De ex, daca avem o clasa intr-un fisier extern, aceeasi clasa in <head> si tot aceeasi definita cu atributul style=””, se vor lua mereu proprietatile din ultima din ele
* Totusi, datorita mostenirii, se vor combina proprietatile din toate 3 CSS,doar ca daca se intalnesc unele si aceleasi proprietati, ultima CSS le va supascrie pe restul

**Ordinea**

1. Ce se gaseste in style=”” mereu e luat
2. Dupa style=””, orice id style are prioritate, de ex #uniD{ }

<div class=”test” id=”unId”>Test</div>

si CSS:

.test { color: red}

#unId{color: pink}

Culoarea finala va fi pink

1. Apoi, la urma vin deja clasele
2. Ultimul e numarul de elemente in CSS style definite.Daca sunt clase care ar influenta acelasi element, stilul cu mai multe elemente sunt are prioritate mai mare

Deci, daca avem:

<style>

header.nn div{

color: red;

}

div.test{

color: green;

}

</style>

<header class=nn>

<div class=test>This is real time online HTML Editor</div>

</header>

Culoarea va fi red, deoarece primul style are 2 elemente simple si o clasa(header apoi nn apoi div) si urmatorul are 1 si o clasa in ea(div si test) si de aceea primul e mai important, caci are 2 elemente pe langa clasa. Doar sunt 2 clase ce influenteaza div si trebuie cumva sa se decida care din ele va fi luata

Dar:

<style>

header nav div{

color: red;

}

div.test{

color: green;

}

</style>

<header >

<nav>

<div class=test>This is real time online HTML Editor</div>

</nav>

</header>

Acum textul va fi green, deoarece header nav div nu are nicio clasa specificata, si desi are 3 elemente ,nu e nicio clasa, si al 2 are prioritate ca are clasa, desi are mai putine elemente.

**!important**

* Aceasta va suprascrie orice alte proprietati, indiferent de prioritatea lor

<style>

header nav div{

color: red;

}

div.test{

color: green;

}

div{

**color: yellow !important;**

}

</style>

<header >

<nav>

<div class=test>This is real time online HTML Editor</div>

</nav>

</header>

Culoarea va fi yellow

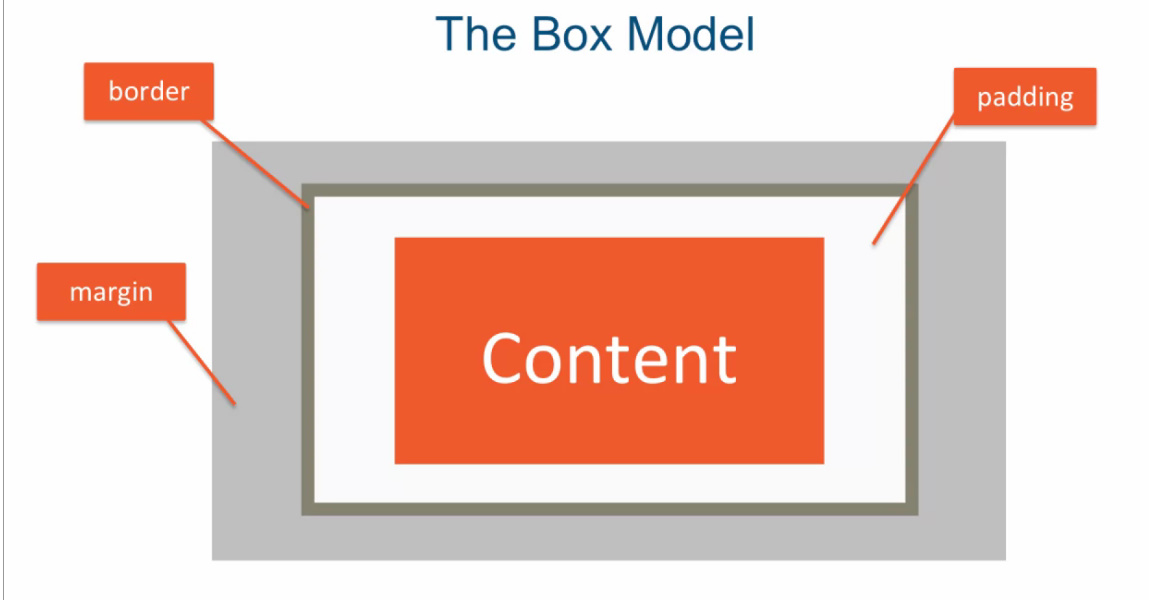
**Text personalisation**

* **font-family** – oferim mai multe fonturi, caci userul s-ar putea sa nu aiba instalat unicul font pe care l-am scrie, de aia dam o lista. La urma mereu punem serif, asa cum neaparat userul are un font din grupa serif instalat

**Size**

* Cea mai buna practica e de a oferi un font-size lui body, pe care orice text ar trebui sa il aiba daca nu are propriile stiluri
* Exista 2 tipuri de marimi pentru size:
* relative
* nerelative
* **Relative**: % si em. Marimile relative nu suprascriu nimic. Ele doar folosesc cele deja existente la momentul folosirii lor si se combina cu ele impreuna.De ex, 2em nu spune nimic browserului, daca nu se stie deja dimensiunea textului.
* %
* em – arata de cate ori mai mare sa fie textul de cat el este la momentul dat. Daca textul nu avea vreun font-size de pe undeva venit la momentul folosirii lui em, se va lua mariema default a acelui font. 2em de ex inseamna ca se mareste dimensiunea textului de 2 ori, 3em de 3 ori si tot asa
* **Nerelative**: px,cm etc.

**Box Model**



* Browserul mereu adauga margin: 8px; la body { }
* width nu arata dimensiunea propriu zisa a elementului
* Daca de ex avem un element:

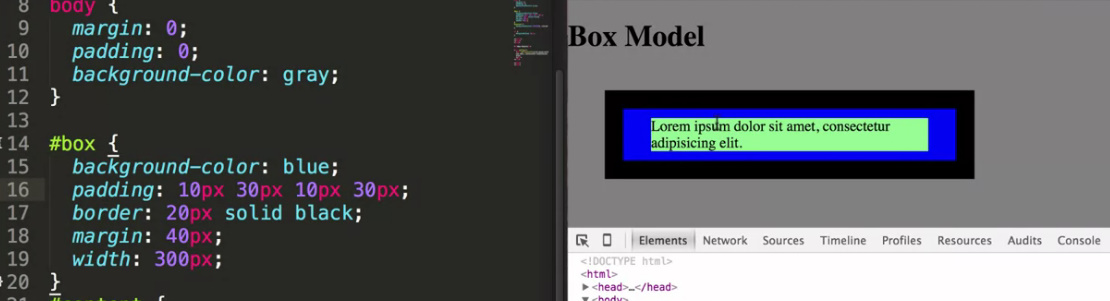
width: 300px;

margin: 20px;

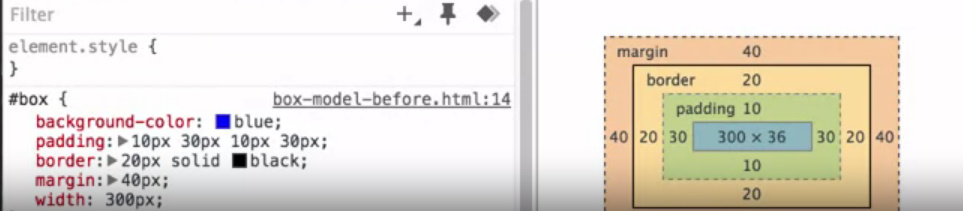
padding: 25px;

Lungimea lui nu va fi 300px;

Lungimea continutului va fi de 300px, dar nu a intreg boxului, adica a casutei elementului. Margin si padding muta continutul, dar boxul elementului continua sa existe mereu. Deci, lungimea boxului va fi 300+2\*20+2\*25





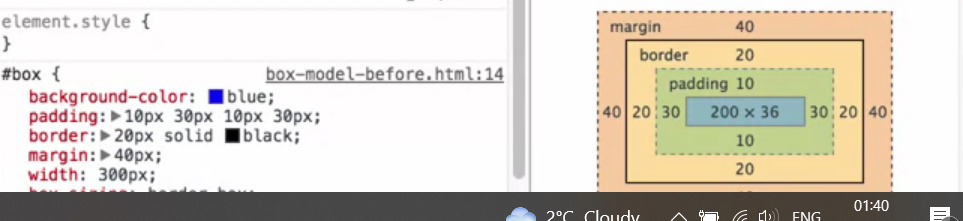


* Daca vrem ca width sa arate nu doar lungimea continutului, ci chiar a intreg boxului, cu tot cu padding si margin, folosim:

**box-sizing: border-box;**

**content-box e default.Bootstrap mereu foloseste border-box**

**Avem grija**

****



**Atentie!Box-sizing nu se mosteneste!**

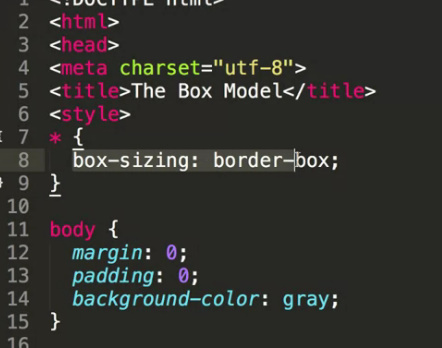
**Atentie si cum se iau lungimile!200+2\*30+20\*2.Nu se va lua in calcul si margin, asa cum el oricum nu face parte din continut, ci muta continutul.**

Dar, trebuie cumva sa il adaugam la toate elementele din pagina, insa nu putem sa il punem la body{}, asa cum el oricum nu va fi mostenit. Pentru asta folosim solutia:

* \* {

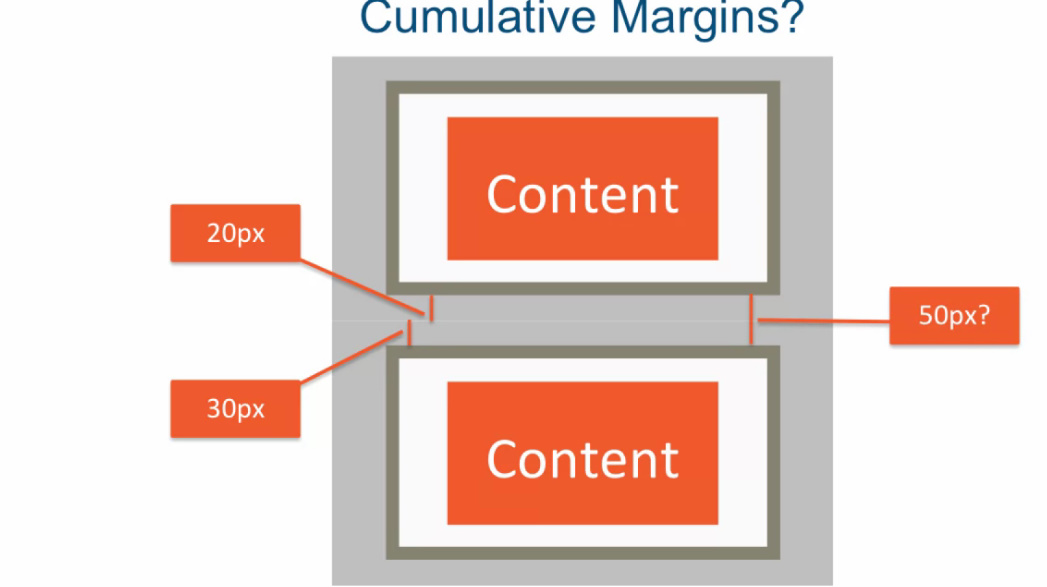
box-sizing: border-box;

}



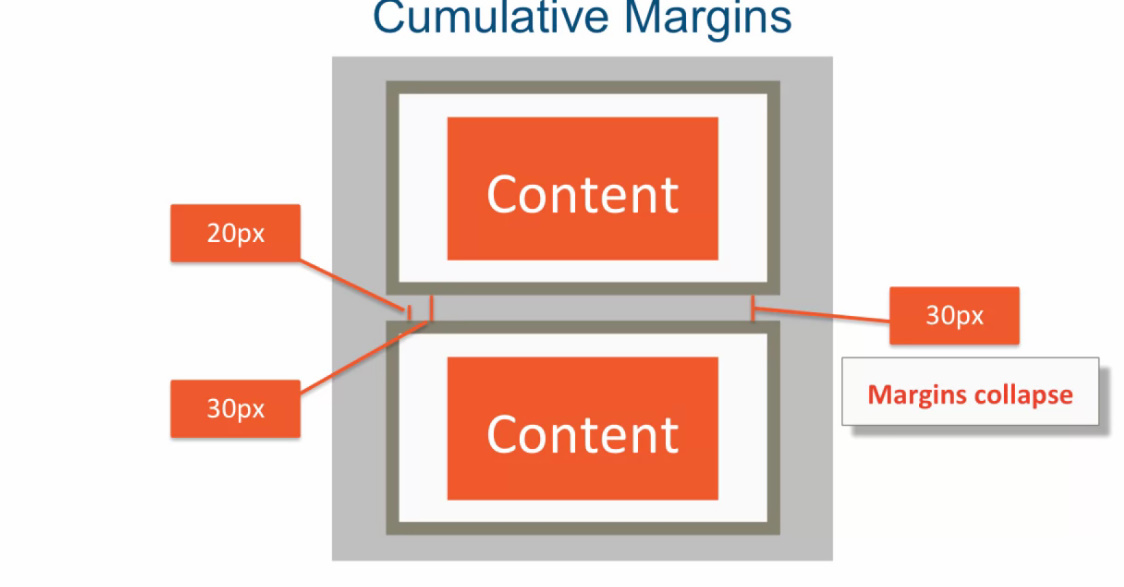
Steluta spune ca orice element va avea asa proprietatea

* Acum, daca avem un element1 si un element2, si element2 e sub element1, marinile lor nu se vor aduna.



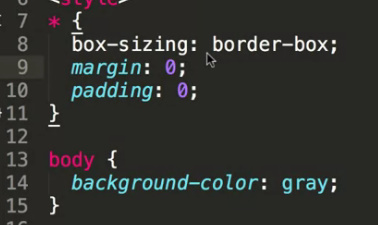
Dar distanta dintre elemente nu va fi 50!

Cel de sus doar vrea o distanta de 20px fata de cel de jos, si ce de jos una de 30px fata de cel de sus.Deci, e logic ca **se va lua cea mai mare distanta dintre cele 2**, deci 30px,asa se respecta conditia pentru ambele. Asa, distanta dintre ele se respecta, nu e necesar sa o adune,



Deci, daca la elementul de deasupra am modifica marin-bottom din 20 in 25 de ex, nu s-ar modifica nimic.

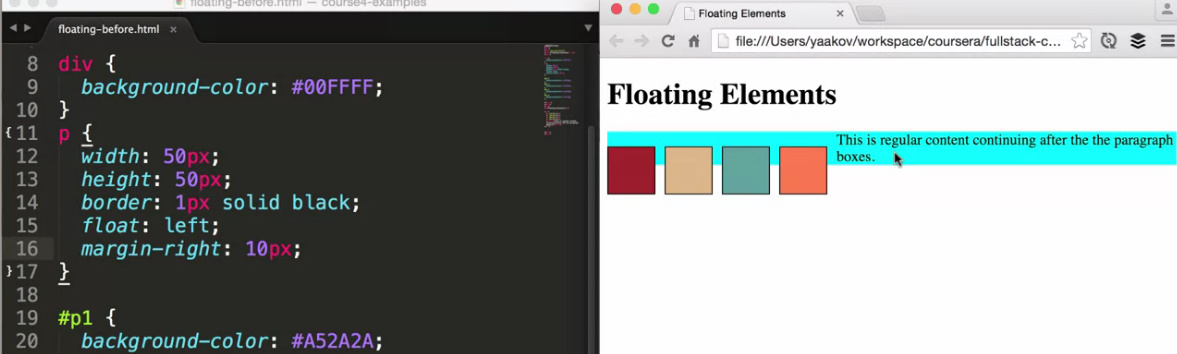
* Browserele pot adauga valori default pentru unele elemente pentru margin si padding, dar asta o putem rezolva cu \* { }

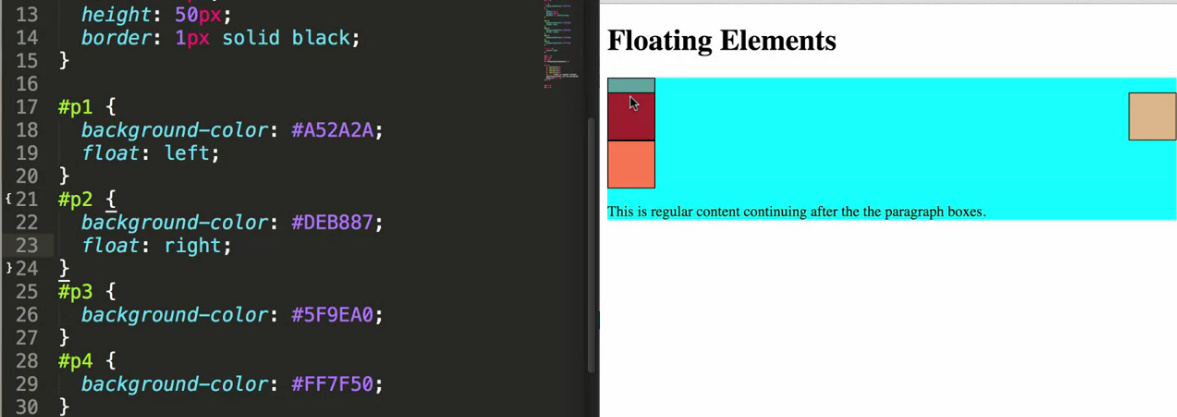


**Background**

* **background-position**: right top; - specifica pozitia la background din element
* **background**-**position**: right; - il pune automat in centru, la dreapta, dar putem scrie center

**Float**

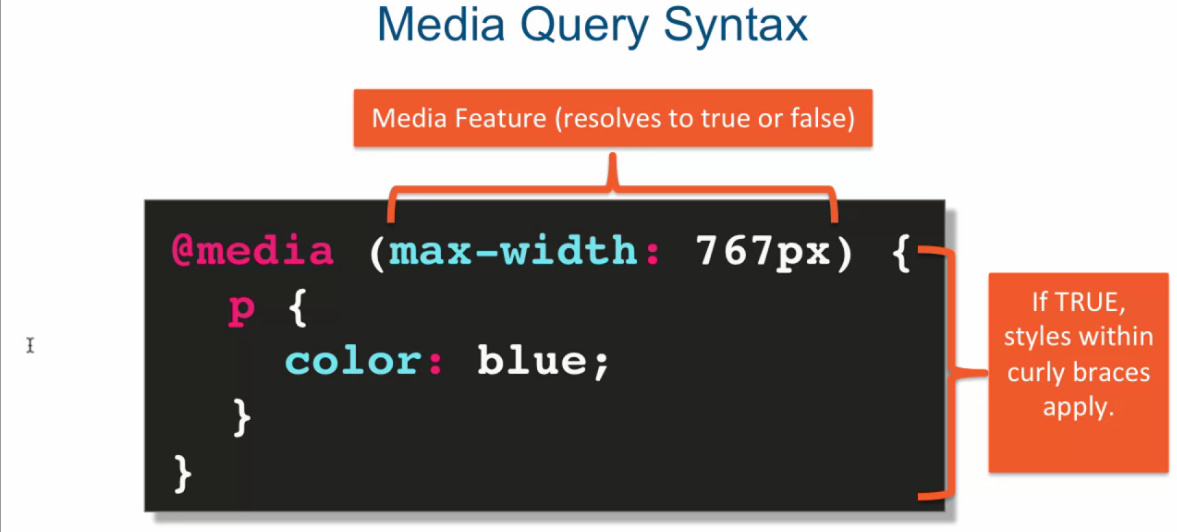
* Proprieatea float face ca un element sa fie ca si cum eliminat din blocul in care se afla, iata de ce de ex un div nu va fi redimensionat dupa un alt div din el ce are float
* 

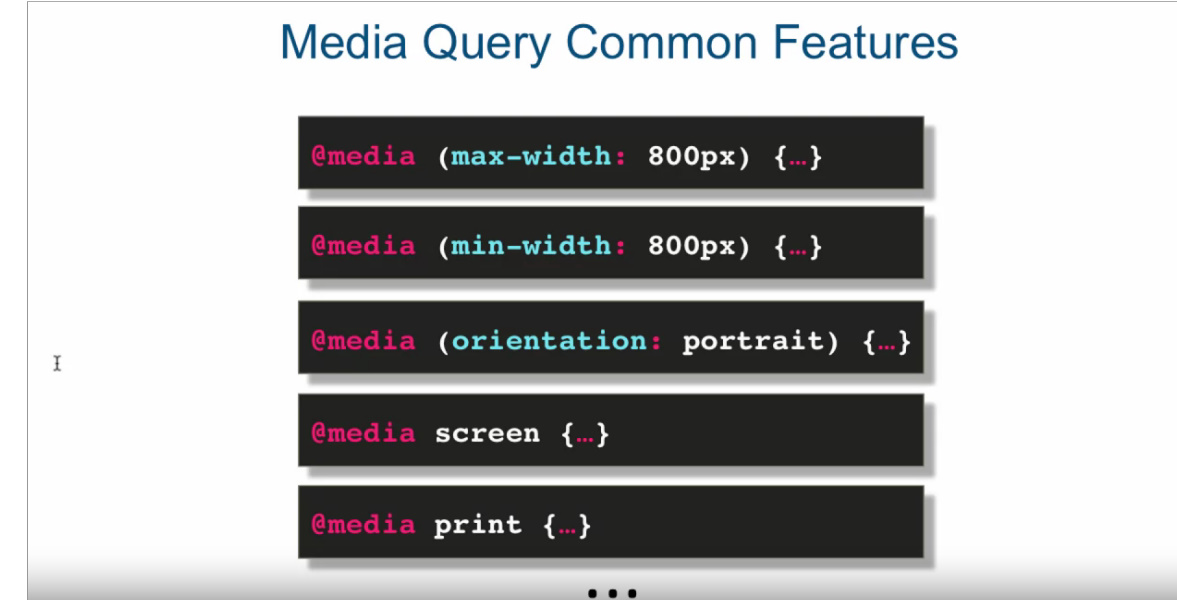


**Absolute**

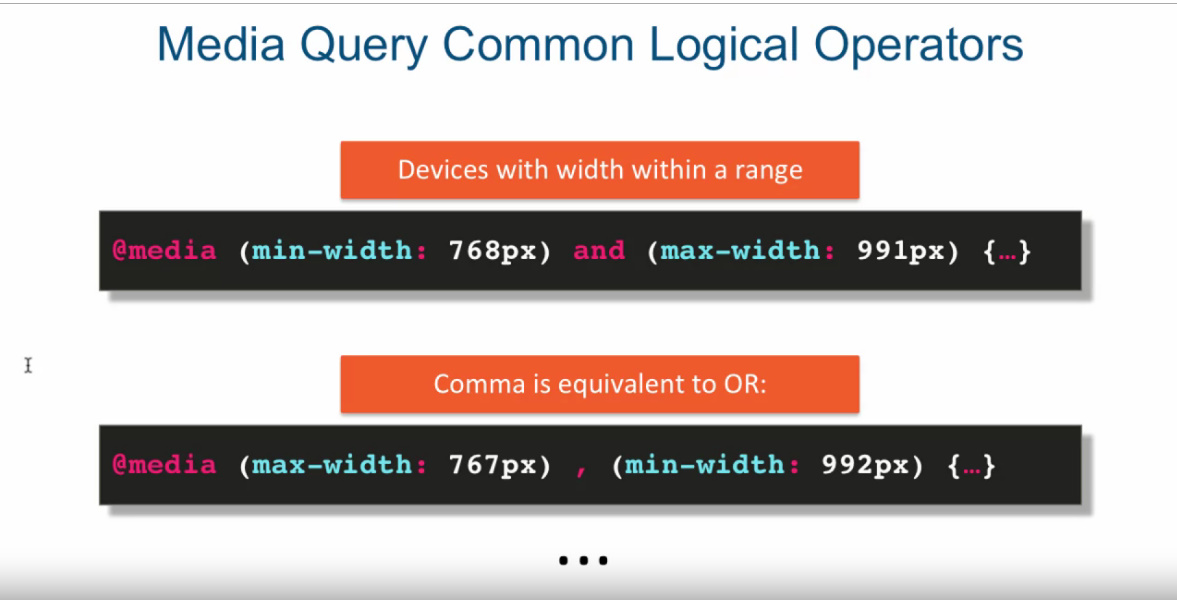
* Pozitia sa este mereu luata in raport cu un parinte absolute sau relative. Cauta elementele partinte pana da de unul dintre ele
* <html> are relative automat nu si body

**Media query**





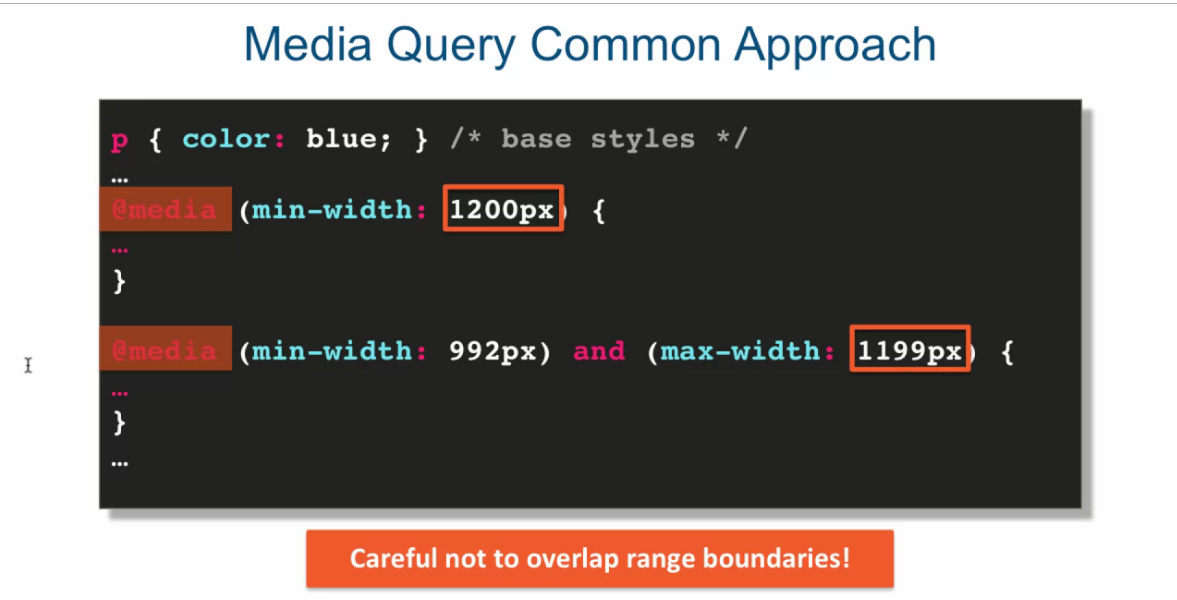
Totusi, max/min-width sunt cele mai utilizate



virgula este echivalenta cu OR



Trebuie neaparat sa avem un stil de baza de la care sa porneasca toate media query, apoi sa adauge ceva de la ele sau sa modifice ceva deja existent

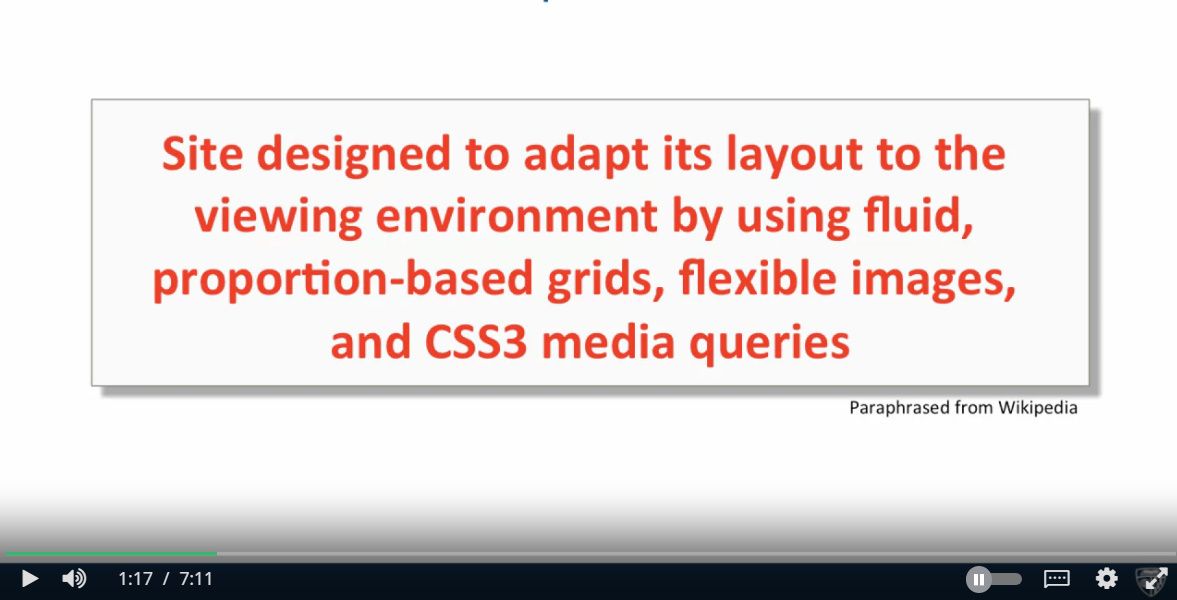


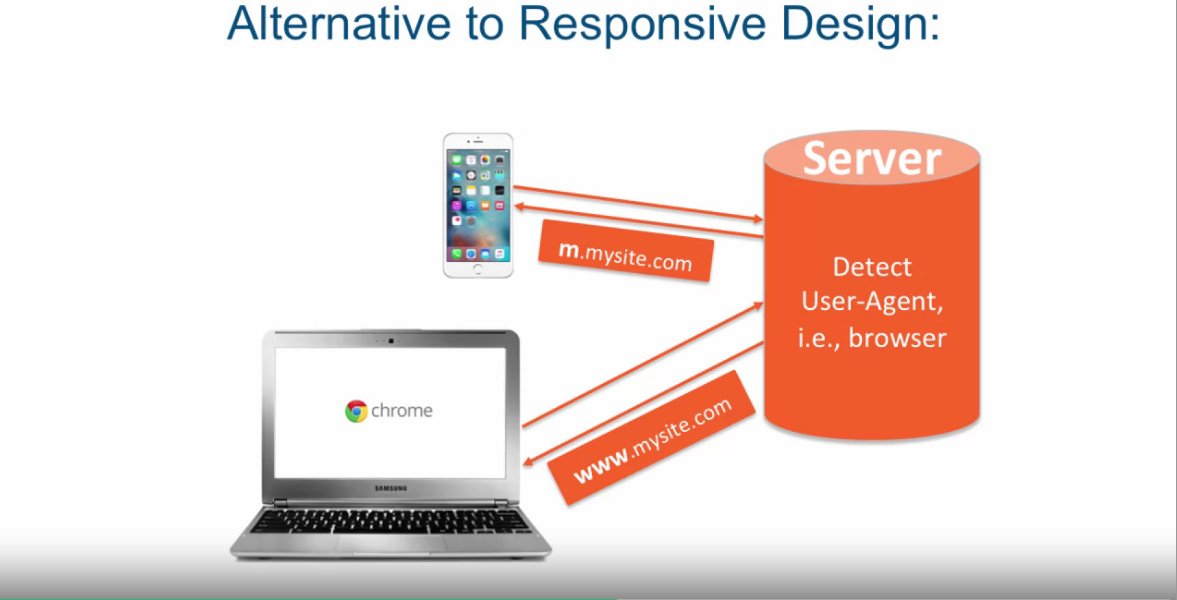
Mare grija sa nu amestecam intervalele. Daca se amesteca, se vor aplica stilurile din mai multe media query, si atunci comportamentul elementelor poate fi total neasteptat/.

Conventional se ia asa:

* large device 1200px+
* medium device [992,1199]
* small 991-

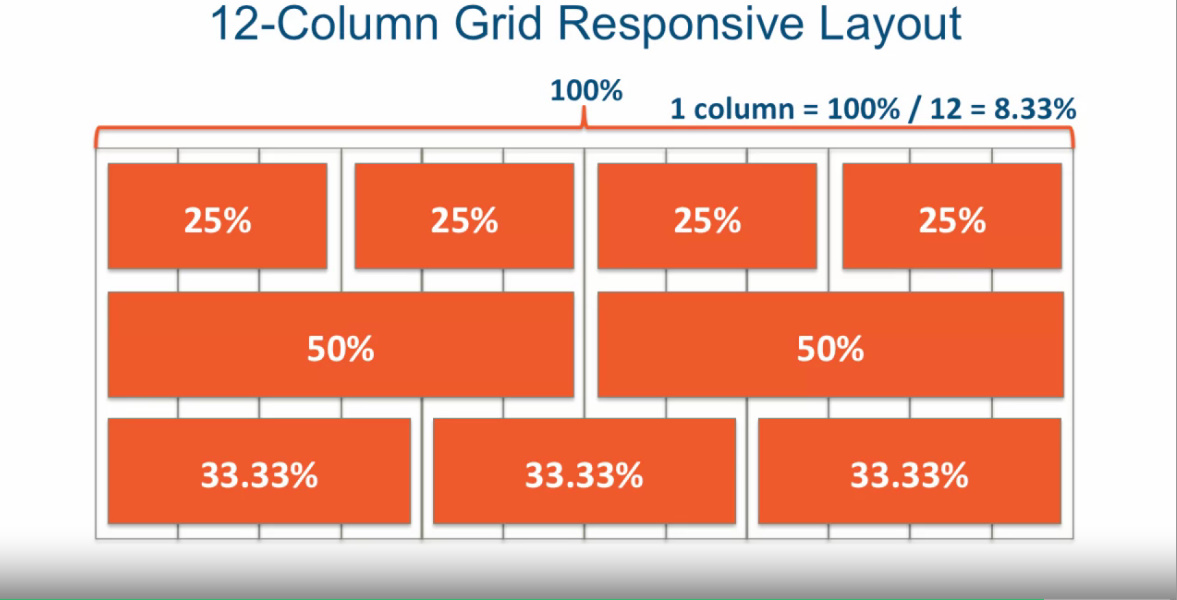
**Responsive Design**



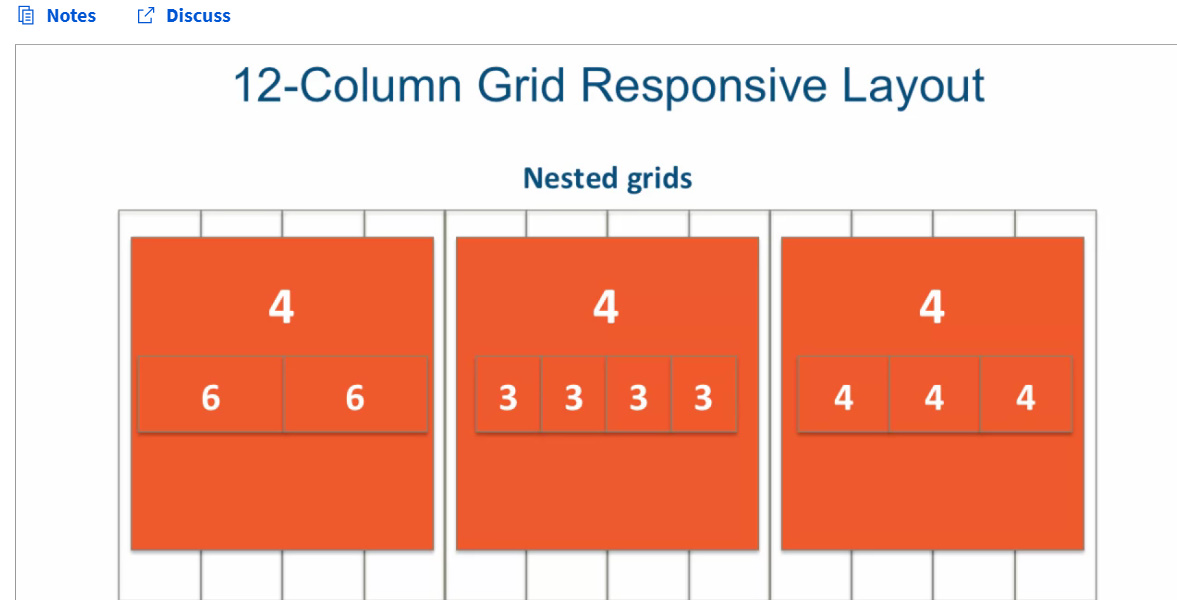


Deci, o varianta ar fi sa facem siteuri separate pentru mobil si desktop, dar problema e ca mobile devices sunt foarte variate, si iar nu putem sa facem una generala. In plus, am avea 2 aplicatii separate.

**12-Column Grid Responsive Layout(aspect adaptabil la grila cu 12 coloane)**

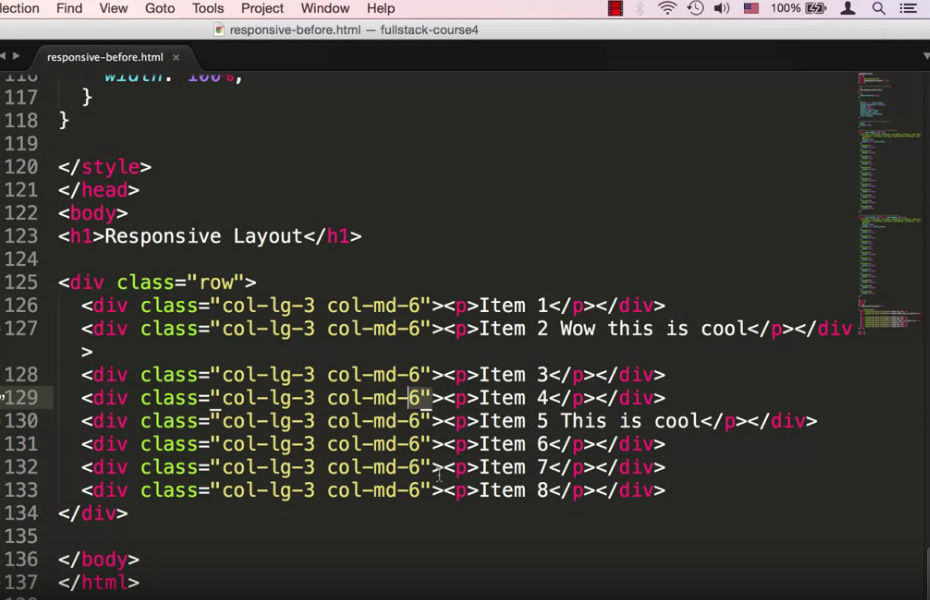


* Tehnica data se bazeaza pe impartirea ecranului in 12 coloane.
* 12 anume deoarece si 1,2,3,4,6 sunt divizori, si deci putem divide in multe parti



Putem avea si layouts in layout

* Este o idee buna de a crea clase diferite in fiecare media query, care fac tot acelasi lucru, nu sa le suprascriem pe cele deja existente, dar depinde de caz, de ex :



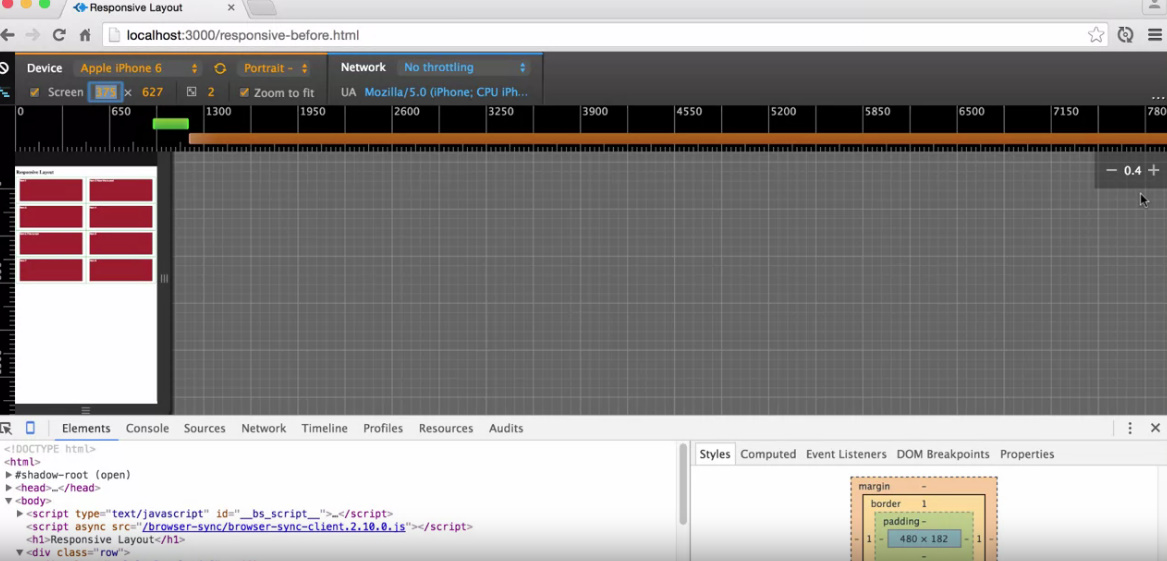
lg-large

md-medium

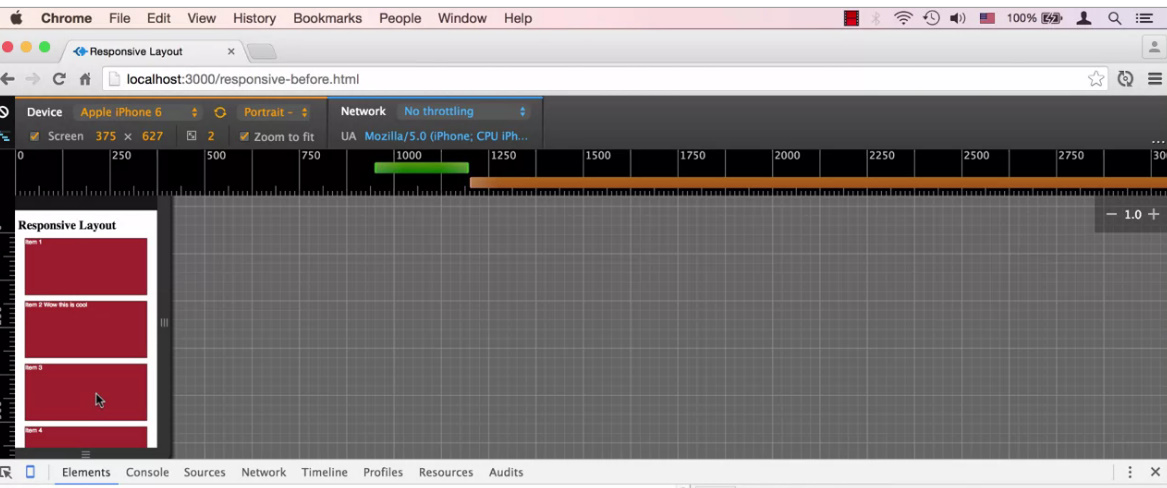
Desi ambele clase raspund de marimea la div, ele se numesc diferit si sunt in media query diferite

* <meta name=”viewport” content=”width=device-width, initial-scale=1”>

Acest tag e foarte important, deoarece browserele mobile incearca sa afiseze tot continutul la site, de aceea ele nu au un zoom de 1, ci de 0.4 deobicei, si asta va face ca dimensiunile elementelor sa nu se comporte asa cum e prevazut,deoarece nu se ia dimensiunea reala la telefon. Asta fac browserele mobile cand siteul nu le spune ca sunt adaptabile.Acest meta va spune browserului ca siteul e responsive si ca sa ia dimensiunea reala la telefon(width=device-width)si poate sa puna zoom 1.Zoomul modifica dimensiunea si o mareste deci.









* **Pentru a putea obtine un site responsive, folosim %, nu px!!!**