**ID và Class trong CSS selectors**

CSS Selectors: là cách gọi được tới các elements (phần tử) trong

html để css

ID (identify): Định danh 1 đối tượng duy nhất

<h1 id = “1”> hi </h1>

~~<h1 id = “1> cc </h1> : không được trùng id~~

Khi select tới đối tượng sử dụng id thì sử dụng dấu #

#1 {css}

Class: Khi sử dụng thuộc tính css chung cho nhiều đối tượng khác nhau

<h1 class = “2”> hi </h2>

<h2 class = “2”> hi </h2>

<h2 class = “2”> hi </h2>

Khi select tới đối tượng sử dụng class thì sử dụng dấu .

.2 {css}

**Mức độ ưu tiên trong CSS**

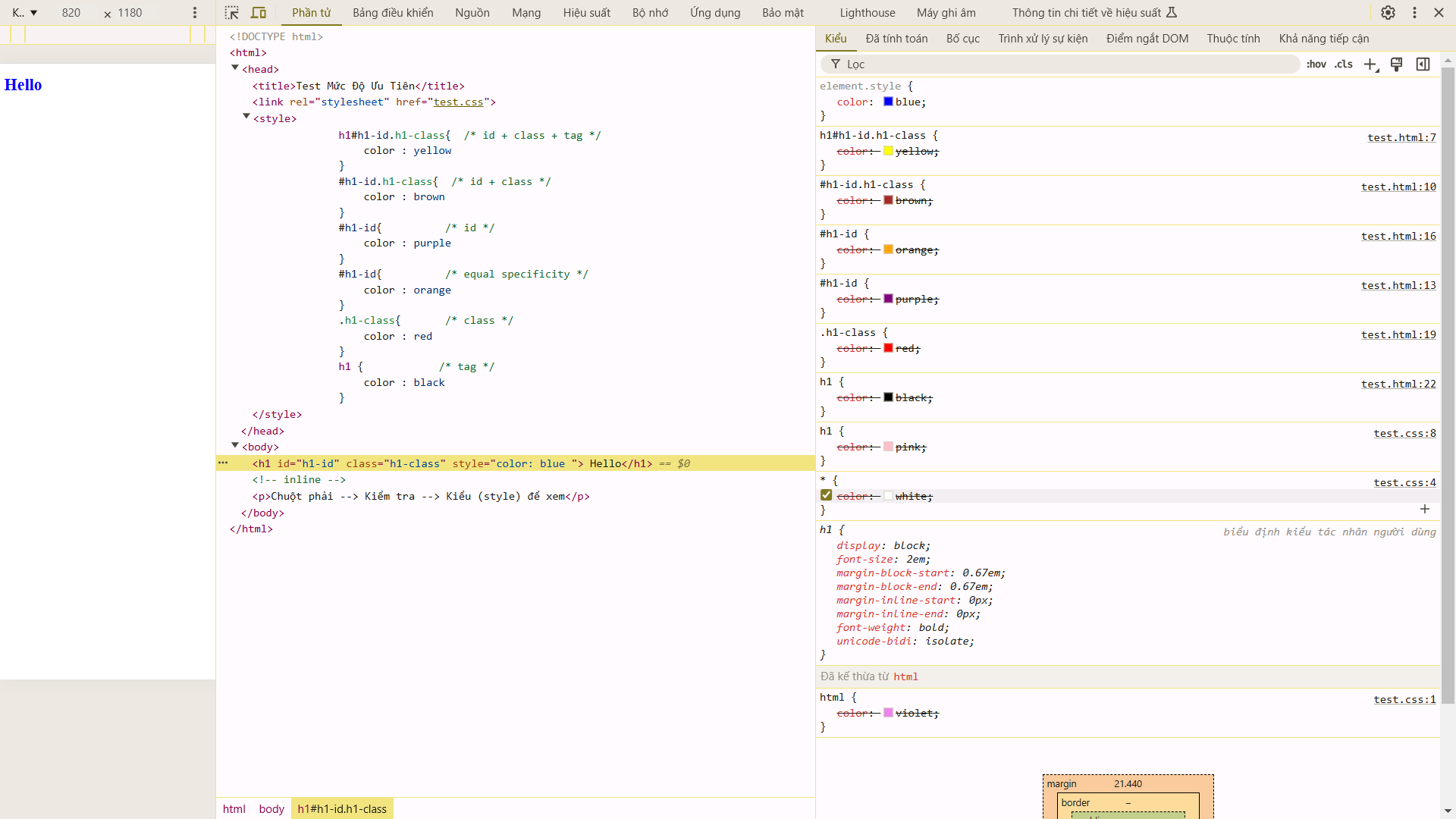
Priority : (ưu tiên)

1. Internal, External : Ưu tiên giống nhau, cách nào được gọi sau thì css đó được ưu tiên hơn
2. Inline
3. #id

Độ ưu tiên giảm dần

1. .class
2. tag (gọi thẳng thẻ h1 {css})
3. Equal specificity ( gọi selector giống nhau, lặp lại)
   1. Universal selector ( \* {}) : phổ cập chung cho tất cả thẻ
   2. Inherited: thừa hưởng (kế thừa) (html {})

Kiểm tra phần style trên web để xem phần thuộc tính ko được ưu tiên bị gạch đi



**!important**

**CSS Variable**

CSS Variable : Kỹ thuật đặt biến trong css

2 kỹ thuật: global và local

* Global:

Sử dụng lớp giả : :root{ }

Cách đặt biến: Sử dụng --tên biến : giá trị (đặt trong lớp giả)

(1 lớp giả chỉ tác dụng đối với 1 biến)

Sau đó gán biến cho thẻ bằng cách sử dụng css function – var

h1 {color : var(--tên biến)}

Công dụng của biến : để chỉnh sửa nhanh chóng bằng cách chỉ chỉnh sửa 1 nơi là ở biến

* Local : đặt biến ở trong thẻ luôn

h1 { --tên biến: giá trị;

color : var(--tên biến)}

**CSS Units**

CSS Units : là đơn vị được sử dụng trong CSS để giúp điều chỉnh kích thước của những đối tượng hiển thị trên web.

* Absolute Units (Đơn vị tuyệt đối)
* px (pixel):
* 1 px trong css tương ứng với 1 điểm ảnh trên màn hình ( với độ phân giải thấp)
* Còn đối với độ phân giải cao sẽ chiếm nhiều điểm ảnh hơn
* Thường sử dụng với thuộc tính width- height …

Ví dụ : Tạo khối:

html : <div></div>

css : width : 100 px

height : 100 px

background-color : red

Đặc tính của đơn vị tuyệt đối: kích thước sẽ cố định, không thay đổi do các yếu tố xung quanh tác động lên ( như kéo giản cửa sổ)

* Relative Units (Đơn vị tương đối): cần 1 nơi để phụ thuộc vào
* % (percent): 100 % = 16 px

Ví dụ : Tạo khối:

html : <div></div>

css : width : 50%

* Phụ thuộc vào thẻ chứa nó là thẻ body
* Thẻ body kế thừa chiều ngang của trình duyệt
* Nên thẻ body nó có chiều ngang = chiều ngang trình duyệt
* 50 % bằng nửa kích thước của thẻ body
* Khi tăng kích thước trình duyệt thì chiều ngang của khối sẽ tăng theo

Khi định nghĩa 1 đối tượng sử dụng giá trị tương đối thì đối tượng này sở hữu 1 kích thước không cố định, sẽ bị thay đổi kích thước khi đối tượng phụ thuộc thay đổi

* Rem (root em): phụ thuộc vào thuộc tính được định nghĩa ở thẻ html

Ví dụ

css : html {font – size : 40 px}

h1 {font – size : 1 rem}

* h1 có font – size là 40 px
* em (percent): phụ thuộc vào thẻ gần nhất chứa nó có thuộc tính liên quan

Ví dụ:

html :

<body>

<div style = “ font – size : 50 px”>

<h1>Hello</h1>

</div>

</body>

css : body { font –size : 100px}

h1 { font – size : 1 em}

* div chứa h1 và gần nhất nên h1 có font – size : 50px

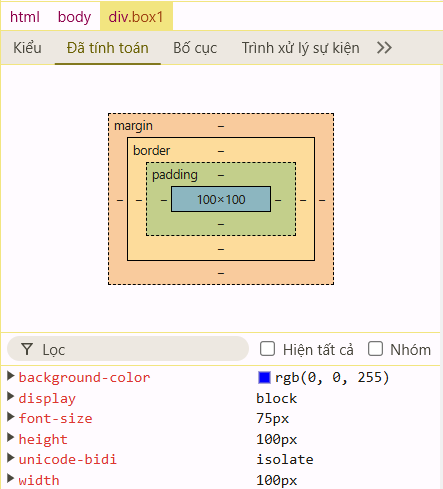
Thường sẽ sử dụng rem

* vw - vh (viewport) : khung hình của trình duyệt, không phụ thuộc vào thẻ chứa nó như %

1 vw = là 1 % chiều ngang khung hình

**Padding**

Padding: vùng đệm thêm



css : .box1{ width :100px

height : 100px}

html: <div class=”box1”></div>

100x100 (số này do css) : lớp nội dung, lúc này chính là kích thước của div (lớp nội dung và div là khác nhau)

Padding : lớp đệm ( có 4 hướng, dấu - là 0)

Khi thêm thuộc tính đệm thì kích thước của div tăng lên nhưng kích thước content ko đổi (div = padding + content)

Ví dụ : css: .box1 {padding-top: 10px}

* box1 : 100x110 , content: 100x100

Viết gọn : 1 giá trị : padding :10px ( trên dưới trái phải)

2 giá trị : padding : 10px (trên dưới) 20px (trái phải)

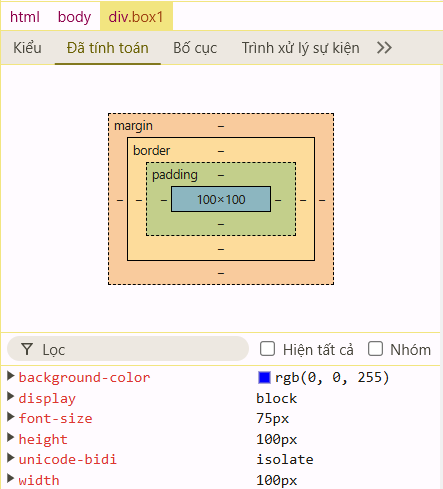
3 giá trị : padding : 10px (trên) 8px (phải trái) 20px (dưới)

4 giá trị : trên phải dưới trái ( cùng chiều kim đồng hồ)

**Border**

Border (nét liền): Lớp viền ở ngoài cùng ôm 1 elements

Margin : ko hẳn là ôm đối tượng nên k được ký hiệu là nét liền



css : .box1{ width :100px

height : 100px}

html: <div class=”box1”></div>

Thuộc tính : Độ dày border-width ( viết gọn)

Kiểu border-style : dashed (nét đứt), solid (nét liền)

inherit (kế thừa), initial (gtri ban đầu), none (hủy border

Màu border- color

Khi thêm thuộc tính đệm thì kích thước của div

Viết dài : border-top-width

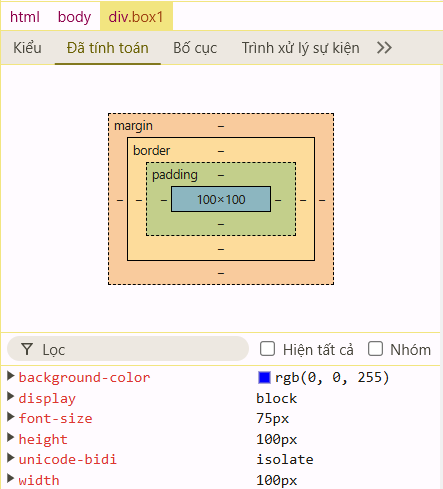
Mặc định style solid sẽ có width = 2 px

Viết gọn : border: width style color (4 hướng)

border-top: width style color ( có thể bỏ thuộc tính)

**Margin**

Margin : Giải quyết vấn đề mà lớp border & padding ko làm được là tạo khoảng cách giữa các đối tượng (border & padding chỉ làm dày bản thân)



Khi thêm thuộc tính margin thì ko làm tăng kích thước của div (lý do tại sao k được hiểu là lớp bao bọc toàn bộ elements, mà nó là chỉ là khoảng cách đẩy ra từ elements)

Để hiển thị theo chiều ngang bên trái float : left

Thuộc tính: Margin-top

Viết ngắn: y chang padding

**Tóm lại:**

Padding: để tạo khoảng cách giữ lớp context và lớp ngoài cùng (border)

Border: để tạo viền

Margin: Tạo khoảng cách cho đối tượng

**Box-sizing**

Box-sizing : Giải quyết vấn đề mà lớp border & padding gây nên làm thay đổi kích thước của elements, làm cho kích thước trở lại như cũ

Thuộc tính: box-sizing : border-box

Khi đó chiều ngang bằng tổng kích thước của border, padding, content

Border, padding sẽ giữ nguyên

Content sẽ được tự động tính toán: Kích thước khai báo – border – padding

Nếu kích thước b & p quá lớn thì content = 0 và elements tăng lên

Thường apply vào base CSS.

Mặc định : box-sizing : content-box (ko làm gì)

Hủy : box-sizing : unset

A black screen with white text

Description automatically generated

**A blue square with black text

Description automatically generated**

**Background-clip**

Background-clip : Để quyết định việc background color được đổ từ ranh giới nào ( từ border, padding, content)

Mặc định background-clip: border-box ( Đổ từ border)

A screenshot of a computer

Description automatically generated

background-clip: padding – box ( Đổ từ padding)

background-clip: content – box ( Đổ từ content)

A blue and green square with black text

Description automatically generatedA blue and green square with black text

Description automatically generated

**Background-image**

Background-image: dùng để đặt ảnh nền

Gắn url hình ảnh (đưa ảnh trực tiếp) : background-image: url(“link”)

Điều chỉnh: background-size : 100px (width: 100px, height : auto)

Khi đó hình sẽ có width là 100 px, height sẽ tùy chỉnh sao cho tỷ lệ giống với tỷ lệ của width

Có thuộc tính mặc định là repeat (lặp lại ảnh) vì ảnh sau có kích cỡ khác với kích thước ảnh ban đầu đã chiếm chỗ

Bỏ đi lặp lại background-repeat : no-repeat

Lặp lại theo trục repeat-x ; repeat-y

Đặt nhiều ảnh nền: url, url, ảnh nào viêt trước thì lên trên

Đưa ảnh= function của css– linear-gradien(dải màu chuyển đều đặn)

Function có tham số: linear-gradien (0, red, pink)(Mặc định 180deg)

Hướng xoay của màu chuyển : 0 -> 360deg ( có -90deg)

Màu bắt đầu chuyển

Màu điểm cuối chuyển

Kết hợp hình ảnh với màu chuyển

Cách chuyển thành màu trong suốt(apha) : rgba (red, green, blue, apha)

Mã màu #xxx gọi là thập lục phân

Cách lấy số này trong devtools: bấm thẳng zô màu trc phần style để hiện lên bảng mã màu

Đổi thành màu trong suốt bấm mũi tên và chỉnh apha (0🡪1)(0 là trong suốt)

Linear-gadient(0,rgba(255, 255, 0, 0.5), rgba(0, 0, 255, .5)

Ứng dụng sự lặp lại: sử dụng ảnh có nhiều chi tiết giống nhau

**Background-size**

Background-size: Thu nhỏ hình ảnh nếu ảnh quá lớn.

Background-size : 100px (width: 100px, height : auto)

Khi đó hình sẽ có width là 100 px, height sẽ tùy chỉnh sao cho tỷ lệ giống với tỷ lệ của width

Sử dụng keyword thay vì sử dụng giá trị tuyệt đối/ tương đối

Background-size: contain

Tính chất hoạt động của contain: lấy chiều bức ảnh dài nhất (dài or rộng) nhưng phải đảm bảo yếu tố là không bị che khuất hoặc không bị cắt xén 1 phần hình ảnh, chấp nhận khoảng trắng

Ví dụ có 1 ảnh 1920x1280 (3:2), nó sẽ lấy chiều rộng 1920, sẽ hiển thị ảnh trọn vẹn, không bị khuất, có kích cỡ phụ thuộc vào chúng ta cài đặt. Khi tăng chiều rộng đến 1 mức mà height vượt quá kích cỡ mà chúng ta set thì sẽ ngừng k phóng to nữa. Nếu tăng tiếp sẽ xuất hiện khoảng trắng bên cạnh ảnh

Background-size: cover

Tính chất hoạt động của contain: lấy chiều bức ảnh dài nhất (dài or rộng) và chấp nhận bị che khuất hoặc bị cắt xén 1 phần hình ảnh, không chấp nhận khoảng trắng.

Ví dụ có 1 ảnh 1920x1280 (3:2), nó sẽ lấy chiều rộng 1920, sẽ hiển thị width không trọn vẹn, bị khuất, chỉ thể hiện 1 phần (nếu ảnh lớn) nhưng height sẽ đủ. Kích cỡ ảnh sẽ phụ thuộc vào kích cỡ chúng ta set. Nếu ta hiển thị hết chiều rông, nếu kéo tiếp thì height sẽ phóng to và bị khuất ảnh

**Background-origin**

Background-origin: tương tự với background-clip nhưng đối với background-image

Mặc định background-origin: padding-box

**Background-position**

Background-position: Để tùy chỉnh vị trí hình ảnh nền trong CSS

5 keyword vị trí :top right bottom left center

Ví dụ : Background-position: top right

Background-position: Bottom (Nó sẽ hiểu là Bottom Center)

Background-position: 50px (Cách lề trái 50px và Center)

Sử dụng keyword + Giá trị:

Background-position: top 20% right -20px (Cách trên 20% và dịch thêm phải 20px )

Sử dụng 2 giá trị: Background-position: x y

Với mặt phẳng tọa độ có gốc (0;0) là sát trên bên trái

Background-position: 50 100 (dịch phải 50px và dịch xuống 100px)

**CSS Background ShortHand**

Background shortHand: cú pháp ngắn gọn

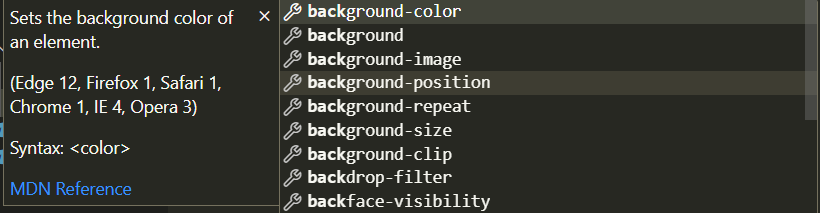
Background-color : red 🡪 Background: red

Background-image : url 🡪 Background: url

Background: image repeat position / size

Background: url no-repeat center / contain

Để hiểu hơn về thuộc tính CSS, ta trỏ vào phần gợi ý của từng thuộc tính và click MDN Reference 🡪 Open và lướt tới phần Syntax



**CSS Functions**

* var(): đặt biến (CSS Variable)
* linear-gradient(): dải màu

(Background-image)

* rbga(): thông số hex của màu và apha màu
* rbg(): thông số hex của màu
* calc(): tính toán
* attr(): lấy thuộc tính attribute của thẻ

rgb() : background : rgb(190, 124, 157)

calc(): width: calc(120px + 40%)

calc(): width: calc(120px + 40%)

attr(): Thường dùng với phần tử giả

html :<a href = “url”>Click vào link này <\a>

Trường hợp normal : css : a::after { content: “fdfsdfsd” }

Kết hợp attr : css : a::after { content: attr(href) }

* Lấy content của attribute href của a sau đó điền vào nội dung đó vào cuối câu thông qua lớp giả after

Trường hợp nối thêm từ thì cần phủ bên ngoài dấu nháy kép: “từ“

css : a::after { “từ” content: attr(href) }

**CSS pseudo-classes**

CSS pseudo-classes: Lớp giả

Cách trình bày : :name\_pseudo\_classes

* :root(CSS Variable): lớp giả này sẽ tham chiếu đến chính phần tử gốc của file html. Phần tử gốc là phần tử bao trùm toàn bộ tài liệu là thẻ <html>
* :hover : là nơi định nghĩa vô số thuộc tính css của phần tử nhưng chúng chỉ được kích hoạt khi lia chuột vào phần tử đó

html: <h1>Hello<\h1>

css : h1:hover{ color: red}

* :active: giống :hover nhưng là click chuột
* :first-child : là nơi định nghĩa vô số thuộc tính css của phần tử để có thể select được thẻ con đầu tiên

html: <ul>

Tập hợp thẻ li đgl tập con của thẻ ul

First-child là thẻ đầu tiên

Last-child là thẻ cuối cùng

<li>first-child </li>

<li>second-child </li>

<li>last-child </li>

<\ul>

css: li:first-child {css}

* :last-child: tương tự first-child
* Thường dùng ở Menu
* Còn nhiều lớp giả khác nữa

**CSS pseudo-elements**

CSS pseudo-elements(Phần tử giả): Tạo ra các phần tử hiển thị trên web mà không cần viết mã html, chỉ cần viết css

Cách trình bày : ::name\_pseudo\_elements

Thuộc tính content ko thể thiếu trong phần tử giả

* ::before : tạo phần tử giả đầu tiên cho phần tử nào đó

html : <div class=”box></div>

css : .box::before { content: “” ( Giúp elements pseudo tồn tại được)

display: block (Hiển thị dạng khối) }

\*Mỗi elements có nhiều pseudo\_elements nhưng phải khác nhau

* ::after: tương tự before
* ::first-letter : chữ cái đầu của phần tử nào đó

html : <h1>Hello<\h1>

css : .h1::first-letter { font-size: 50px}

* ::first-line: Dòng đầu
* ::selection: Bôi đậm

🡪Nếu ko có phần tử ở đầu thì sẽ hiểu toàn bộ phần tử bị ảnh hưởng

**Position**

Position(Vị trí): Để làm được giao diện đè lên nhau

Tạo ra những vị trí, thiết lập những vị trí hiển thị cho các elements trên web, các vị trí rất đặc biệt (đè lên nhau):

* Relative(Vị trí tương đối)
* Absolute(Vị trí tuyệt đối)
* Fixed(Vị trí phụ thuộc vào khung trình duyệt)
* Sticky(Vị trí bám dính \_ Không khuyến cáo)

Relative:

Không phụ thuộc vào đối tượng nào khác

Lấy vị trí đang đứng của nó làm gốc tọa độ

Có thể thay đổi vị trí của nó, kể cả đè lên đối tượng khác

html : <h1>Hi</h1>

css: h1{ position : relative

top : 100px

left : 100px }

Top left chỉ hoạt động khi có position

Absolute: Đối tượng này phụ thuộc vào đối tượng khác

Phụ thuộc vào thẻ cha gần nhất có position làm gốc tọa độ, không quan tâm thuộc tính position là gì.

Có 1 đối tượng là con đối tượng khác và cần di chuyển đối tượng con xung quanh đối tượng cha.

html: <div class=”box”>

<div class = “box-child”> </div>

</div>

css: .box{ position: relative}

.box-child {position: absolute

top: 0 (cách top của thẻ cha = 0)

left: 0 }

Nếu w,h không đáp ứng được độ dài mà thuộc tính có cả:

Top bottom: ưu tiên top, left right : ưu tiên left

Không có w,h mà có cả 4 hướng thì sẽ phóng full

css: .box-child {position: absolute

top: 0

left: 0

bottom: 0

right: 0 }

Fixed: Đối tượng phụ thuộc vào cửa sổ trình duyệt

Ứng dụng: Kéo lên xuống header trang vẫn giữ nguyên

html : <div class =”header”></div>

css: .header{ width : 100%

height: 42px

background: black

position: fixed

top: 0}

Sticky: Khác với Fixed (Fixed sẽ đứng yên khi cuộn, mặc dù chưa chạm top) còn Sticky khi cuộn nếu chưa chạm top khi kéo lên sẽ chạm top sau đó mới đứng yên

html : <div class =”header”></div>

css: .header{ width : 100%

height: 42px

background: black

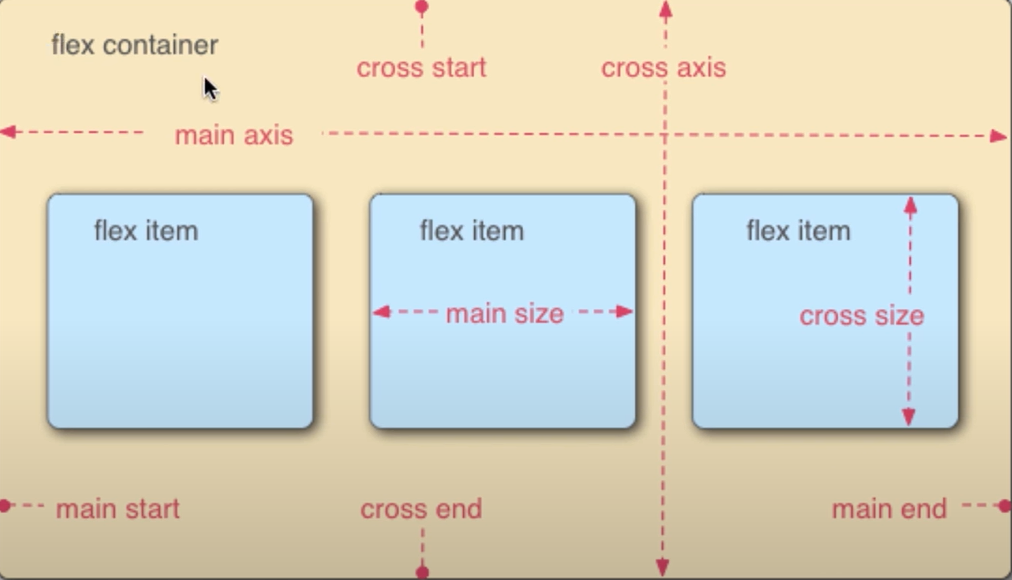
position: sticky

top: 0

margin-top : 50px}

**FlexBox**

FlexBox: Kỹ thuật xây dựng Layout, kỹ thuật có tính linh hoạt rất cao giúp tạo ra giao diện web, bố cục hiển thị trên web



2 thành phần chính : Flex container (Vùng màu vàng) và Flex item (3 ô màu xanh)

Flex Container là thẻ chứa, còn Item là 3 thẻ con trực tiếp của Container

Main axis: Trục chính có điểm đầu điểm cuối là main start & end

Cross axis: Trục chéo luôn vuông góc với main axis, có đầu cuối là cross start & end

Main size & cross size: kích thước của flex item, main size song song với main axis và cross size tương tự.

Chiều vẽ trên hình là ví dụ, thực tế có thể thay đổi được

Mọi giá trị màu đỏ đều có thể thay đổi trong css, đều được cung cấp thuộc tính để dễ dàng tùy chỉnh

Vai trò:

Main axis: quyết định hướng flex item được nằm (ngang, dọc)

Main start & end : quyết định item hiển thị lần lượt theo hướng nào (phải, trái / trên, dưới), phụ thuộc vào trình duyệt

Dồn item sát trái , phải main start/end, giữa main start main end

Trong trường hợp sử dụng trình duyệt đọc ngôn ngữ trái qua phải thì main start nằm bên trái và ngược lại

Cross axit: khi item hiển thị theo nhiều hàng thì sẽ quyết định xuống hàng theo chiều nào ( nhảy trên hay nhảy dưới) phụ thuộc vào cross start.

Dồn item trên, giữa, dưới

Thuộc tính:

Display: flex | inline-flex: quyết định việc có sử dụng layout flexbox hay không, muốn tạo ra flex container, flex item thì sử dụng display

Flex-direction: thay đổi hướng của main axis (mặc định là row (nằm ngang) | column (nằm dọc))

Flex-wrap (Quyết định phần tử bên trong flex-container có được bé dòng nếu không đủ không gian hiển thị trong 1 dòng hay không): Mặc định là nowrap | wrap cho phép item xuống dòng, wap-reverse : thay vì xuống dòng sẽ nhảy lên trên ( đảo chiều cross start & end)

Flex-basis : set kích thước cho main size của flex item <length>,

Main size song song với main axis nên ta tự ứng biến

Justify-content: canh flex-item (trái, phải, giữa) cùng hướng với main axis nằm ngang, còn nằm dọc

Justify-self: Khi set justify-content cho flex container (cha) = set justify-self cho mọi flex item ( con)

Align-content (Tương tự justify-content): canh flex-item (lên, xuống, giữa) cùng hướng với cross axis nằm dọc, còn nằm ngang thì

Align-seft: Tương tự justify-seft

Flex-grow: Thay đổi kích thước của main-size

Flex-shink: ngược lại Flex-grow Nhỏ lại

Flex: short hand của basis, grow, shink

Order: Hiển thị item nào trước (mặc định hiển thị theo main start)

Flew-flow: shorthand của direction và wrap