1. Inline css -> high priority

Inline css -> next

External css -> less.

1. Intrinsic vs extrinsic sizing, width auto vs 100%, min-content, max-content, fit-content.

Default width is auto for any thing. Scroll will not come.

100% with padding or margin scroll will come. Total width will take.

Width: min-content (or) max-content

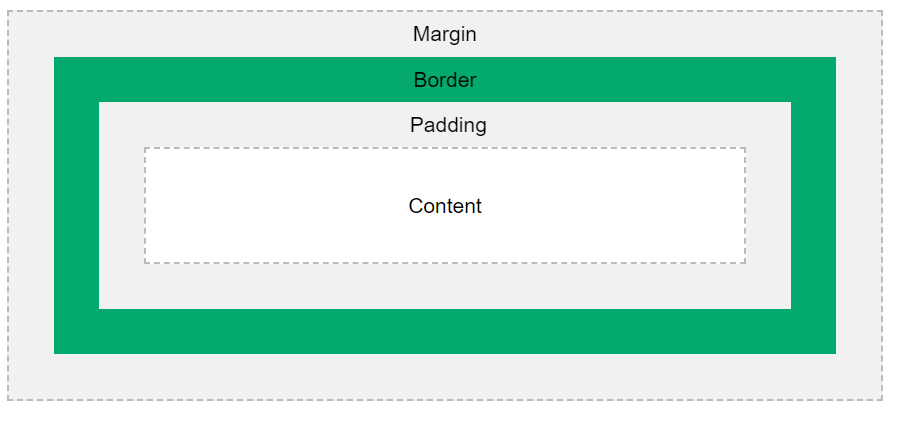
Min-content: shrink the content which content has highest width. When resize window scroll will come.

Max-content: how much content there is it will accept. If content is high it takes scroll.

Fit-conent: same as 100%. And scroll will not come. Based on content.

1. Box-model, content-box vs border-box in box-sizing.

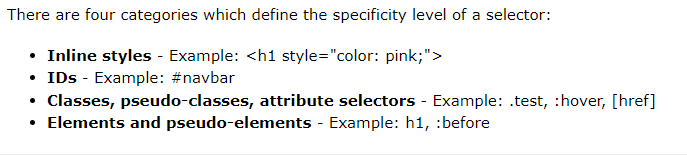
The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model



1. Box-sizing  
   box-sizing:

The default one is content-box. If we use border-box it will not add extra width and height for box-model not allowed.

1. Css selectors



\*{} -> universal selector. (0 points)

P {} -> element selector. (1 points)

.class -> class selector. One or more classes have elements. (10 points)

#id -> id selector. Only one id present in selector. (100 points)

a[href]{} or a[href\*=”google”] -> attribute selector. (\* any were, ^ starting, $ ending) (10 points)

inline (1000 points)

!important (10000 points)

css algorithm will work rules:

* Position and ordering of appearance. The latest one will take.
* Specificity.
* Origin (whether it is from browser or html file).
* ! important

1. Grouping selector, pseudo class and pseudo elements.

h1, h2{ color: red} -> grouping selector. We can also group class and id’s.

pseudo class: -: (:)hover, active, link, focus. (Change state).

pseudo elements: - (: :) new element in css. ::before, ::after, ::marker, ::selection.

1. Descendent selector and direct descendent selector for child element.

Descendent selector -> .myclass p {}.

direct descendent selector

“>” -> .myclass > strong {} for direct children. Not for nested children

“+” ->. myclass + p{} . use for adjacent selector. Only immediate selector

“~” ->. myclsss ~ p{} . subsequent sibling selector. How may times occur it will add style.

“&”->. myclass & p{}. Component selector. If we have same class for div and p. we need only for ‘p’.

Inheritance: -  
 child will inheritance from nearest parent. Margin, padding, border, bg will not inheritance.

If we want Margin, padding, border, bg to children we need to use ‘inherit’. If parent and child we using same above properties we use ‘inherit’. Apply both inheritable and non inheritable.

If we no need parent style elements to particular child classes we can use ‘initial’. apply css default value or initial value

‘unset’ it is domination of ‘inherit’ and ‘initial’. all: unset. If it is inheritable if it not non-inheritable apply default value.

‘revert’ can control ‘inheritance’. Only particular css property. Bg-color: revert.

It takes browser style only. Apply browser applied value not take initial value.

1. Sizzing units

Line-height: 1.5 -> 150%

Absolute length -> cm, mm, q, in, pc, pt, px.

em -> font size of parent element.

rem -> font size of root element.

lh -> line height of parent element.

rlh -> line height of the root element.

1em -> 16px.

1rem -> 16px.

vw -> view port width.

vh-> view port height

1. Display layout methods block and inline block

In olden days we don’t have layout methods we use table elements.

Layout history:-

Css1(1996) -> floats

Css2(1998) ->

Css3(2010) -> responsive web design

2012 -> media queries and flex box.

2017 -> grid

2019 -> intrinsic web design

2021 -> container queries.

Block -> if two div in a div it come in new line. We can set (width and height, margin, and padding).

Inline -> if two spans in a div it will come side by side. We can’t set (width and height, margin, and padding (top and bottom)). To add these properties, we need to use inline-block. If you add ‘block’ it crates spans in same line.

Inline-block -> height, width does not allow. We need to use inline-block to allow h, w, m, p.

Table -> showing div element as table.

Table-row (display) ->

Table-cell (display) ->

1. CSS floats (left, right, clear both). In layout designing.

If two block level element one Image and other paragraph. If we want to show side by side.

Image tag CSS use “float left”. (Image and left side text data will come).

If only one paragraph tag needs to float aside of image. Another paragraph we need to show bellow and another paragraph we need to use “clear: both”.

1. CSS multicolumn layout using column-count, column-width, column-property

Show content newspaper margined type.

If you have container with colums 4 coming in 4 rows. If you want show 4 colums in one row. We must use below one.

.colums{

column-count:4; //it creates 4 columns.

Column-width:250px; // each column min width.

column-rule: 1px solid red // to add border.

Shorthand notation for ‘column-rule’.

column-rule-style: double;

column-rule-width: 1px;

column-rule-color: red;

column-gap: 2em; // default gap 1em. Gap b/w each column

Text-align: justify; // text equally spread.

Column-fill: auto; // if it has height we need to utilize all height which we provided.(default balance)

}

To show blockquote common for all we need to use. Bellow one.

Blockquote{

Column-span: all;

}

// shorthand notation column: 250px 4;

1. Css positioning properties.

Static: - default. Content shows normal flow only.

Relative: when we want to move content from original location left, right, top, bottom we need to use relative. One position to another. Z-index will apply position other than static.

Absolute: the place we are using it will come out of the box (view port). When we add left, right, top, bottom (it will move were ever we want relative to parent). If we use relative css to parent element that time child element move across the parent element.

Fixed: if element we set to fixed. It will remain fixed at that position only. Element Came out of view port. The element remains in the same place when we are scrolling time also.

Sticky: when element and it at middle is sticky with top:0px. Then we scroll if the element came at top 0px it fixed at position. ()

Make css fixed header and footer for web page using position fixed

For header, footer {

Position: fixed,

Width: inherit ,

Max-width: inherit,

}

Header{ top:0px}

Footer{bottom:0px}

Sticky position property in the reel time

H1{

position: sticky,

top:0px,

}

1. Flexbox layout mechanism   
     
   use for laying of group items in one direction.

Features: -

* Display items either row or column
* Writing mode of document
* Default single line we can wrap in multiple lines.
* Space b/w items equally distribute the items in inside and outside.

Flex-direction left to right or top to bottom. Default value “row”.

For block level elements come one by one down.

.container{

Background: red,

Display: flex, // block level element will come single line.

Flex-direction: row, // default row (row-reverse it will move right side and items right to left)

(column, will come one by one)

(column-reverse, items will come reverse)

Flex-wrap: wrap, (default no-wrap) if maximum items in row we will get scroll in display. To not allow scroll we use ‘flex-wrap’, ‘wrap allow next line.’

Flex-flow : column wrap, (we can set flex-flow and flex-direction it is short hand notation).

}

.items{

Background: yellow,

}

Controlling space b/w flex each ‘items’ to write styles. Below three properties we have shorthand notation ‘flex: initial’ (auto, iteam which has large data it take large space otherwise all items same space ). Flex:1 it share all items same space.

Flex-grow: default ‘0’ it will not grow.

Flex-shrink: default ’1’ items shrink smaller than their flex-basis

Flex-basis: default ‘auto’

Flex: 1 -> 1,1,0 (share equal space) flex: auto -> 1,1, auto(based on content it share total width) flex: initial -> 0,1, auto (all without accessing all place).

Flex-basis: -

using display: flex we can’t set width for child elements, so we use this. ex:- 100px when responsive will work.

If we have max-width and flex-basis it will take max-width only. If we have min-width and flex-basis it will take min-width only. If we set width and flex-basis it will take flex-basis.

If it is flex-direction: column it considers as height in parent container.

Flex-grow: -

In container width 300px, two child’s are there each flex-basis: 100px this time. Some space is remaining in container for 2nd child to share that remaining space.

If flex-grow ‘1’ for each child, then two will share same size.

.child: nth-of-type(2){

Flex-grow: 1;

}

Flex- shrink: -  
 if flex-basis: 100px each child when we shrink browser it each child element will decreasing the width.

Flex-shrink: 0; it not to be shrink. 1 means shrink.

If both child ‘0’ that time scroll will come

Short hand :- flex: 1 1 100px (fg, fs, fb)

Rearrange flex items using flex CSS: -

.child: nth-of-type(2){

Order:1 // it will come last.

Order:- -1 // it will come first.

}

Justify-content: - main axis  
 .container{

Display: flex;

Justify-content: flex-start(default), flex-end (all items move to end), space-between (space b/w each item’s), space-around (space will available around). Space-evenly (equal space for all items), center

}

Align-content: - cross axis  
 if we have flex items more than one line in parent.

.container{

Width:80vw;

display: flex,

flex-wrap: wrap,

align-content: default (stretch), center ( in container center will show all lines), space-around( space around each line) space-evenly (all lines with same space distributed), space-between, flex-start, flex-end

}

.child{

Flex-basis: 20%.

}

Place-content: -

Shorthand for Justify-content and Align-content.

.container{

Display: flex;

Place-content: center space-between;

}

Align-items: -

With align-self we will set each Induvial items. With out induvial item we need to set all.

.container{

Align-items: center,

}

Align-self: -

In that based on content it will place in column position. Induvial items

Stretch(default), center (with in the line it is in center), flex-start, flex-end,

.child:nt-of-type(2){

Align-self: center;

}

1. CSS grid layout

Controlling layout provides a two-dimensional layout system, controlling layout in row and columns.

Flexbox we can control either row or colums.

Features:-

* Grid we can define row and columns. We can track row and columns sizes.
* Direct children of the container are placed in grid. We can place the item’s precise location you want.
* Lines and areas on grid can be named to make placement easier.
* Spare space can be distributed b/w the tracks.
* Grid items can be aligned within the area.

For 4 columns layout it has 5 grid lines. Plus 1 adding.

Track is space b/w 2 row lines. a row column track b/w two row lines. A column track b/w two column lines.

Grid cell is smallest place on grid defined by intersection of row and column tracks. It is just table cell or cell in spread sheet.

Grid area it takes multiple tracks it will take.

Tracks gutter or alley b/w tracks. Grid gap.

<div class=”container”>

<div class=” item item-1”>item 1</div>

<div class=” item item-2”>item 2</div>

<div class=” item item-3”>item 3</div>

<div class=” item item-4”>item 4</div>

<div class=” item item-5”>item 5</div>

<div class=” item item-6”>item 6</div>

</div>

.container{

Width:1000px

Bg-color: #eee

Margin-top: 30px

Display: grid // when using display flex all will come side by side. But in grid it will not come.

Grid-template-rows: 150px 150px // it will add two rows for 1st two child elements. We have “repeat(2,150)”. 2\*2 rows, repeat(2,1fr) // for all rows it will take equally height.

Grid-template-columns: 150px 150px 150px // it for columns. // now all will come 3\*3 format. “repeat(3,150)”. “repeat(2,150) 1fr” it will take remaining total space last one. “repeat(3,1fr) ” all three will take equally same space. “50% repeat(2,1fr) ”. “50% 1fr 2fr”. Fr-> fraction units.

Grid-row-gap: 30px //for space b/w each row.

Grid-column-gap: 30px // for space b/w each column.

Grid-gap: 30px // shorthand notation. For row and column gap.

grid-template-areas: "head head head side"

"box1 box2 box3 side"

"main main main side"

"foot foot foot foot";

}

We can use like this also grid lines

 grid-template-rows: [header-start] 100px [header-end box-start] 100px [box-end main-start] 400px [main-end footer-start] 100px [footer-end];

  grid-template-columns: repeat(3, [col-start] 1fr [col-end]) 200px [grid-end];

// using grid-template-area:- naming areas.

.header {

  /\* grid-column: col-start 1/ grid-end; \*/

  grid-area: head;

}

Item-1{ // we need to add for all 6 child items.

Bg-color: red;

Grid-row-start: 2;

Grid-row-end: 3;

Grid-column-start: 2;

Grid-column-end: 3;

Grid-row: 2/3 // shorthand to change positions

Grid-column: 2/3 // shorthand

Grid-area: 1/3/2/4 //shorthand

Grid-row: 2/3 it is used to occupy area in row.

Grid-column:2/4 it is used to occupy area in column.

grid-column: 1/span 3. // span 3 it takes 3 grid items.

grid-row: 1/span 3. // span 3 it takes 3 grid items.

}

Implicit vs explicit grid

 grid-template-rows: 150px 150px;

   grid-template-columns: repeat(2,1fr);

if we are using 2rows and 2 columns. If total items are greater than 4 it automatically shows remaining items implicit grid. And above 4 items are explicit grid.

Grid-auto-rows: 100px; to set height for explicit grids.

Grid-auto-flow: row(default) // column

Align grid items vertically and horizontally using align-items:-

.container{

Align-items: stretch, center, start, end. // only for container

Justify-items: stretch, center, start, end.

}

.item-4{

Grid-row: 2 / span 3;

Align-self: stretch, ; // use for single items.

Justify-self: stretch, center, start, end.

}

.item-7{

Grid-column: 1/-1;

}

Align the grid Tracks in the container.

.container{

Justify-content: center, space, space-around, space-evenly, end // to show whole container items in center. (horizontally )

Align-content: center, start, end, space-between, space-evenly. // vertically

Grid-auto-flow: row dense // like photo gallery.

}