CSS

CSS

- CSS устойчивый (progressive enhancement, фоллбэки)
- декларативный (не точные инструкции, а браузер сам решает как их исполнять)
- контекстный (поток документа и контексты наложения)

Code

Подключение, структура стиля

```
<span style=”color: red;”>Text</span>
```

```
<style> span { color: red; } </style>
```

style.css

```
span {
        color: red;
    }
```

Селекторы (начиная с наиболее быстрого):

```
#id {}
.class {}
div {}
a + div {}
p ~ p {}
ul > li {}
* {}
input[type="text"] {}
a:hover {}, a::before {}
```

Псевдо-классы:

```
:hover, :active,
:first, :last,
:nth-child(), :nth-of-type()
:not(<селектор>), :is/:any/:where(),
:disabled, :focus, :checked, …
```

Скорость обработки селекторов

```
p a {}

p.class {}

p.class span a.class {}

table > thead > tr:first-child th {}

div.class div #id {}
```

Каскадность, наследование, специфичность

Какого цвета будет текст в спанах?

HTML

```
<span class="red blue">раз</span>
<span class="blue red">два</span>
```

CSS

```
.red { color: red; }
.blue { color: blue; }
```

HTML

```
<span id="id" class="class">text</span>
```

CSS

```
#id {
  color: blue;
}
span {
  color: green;
}
.class {
  color: red;
}
```

Специфичность

Селекторы (с наиболее специфичного):

```
#id {}

.class {}, :hover {}, [type=”radio”] {}

div {}, div::after {}
```

Наследование

Все дочерние элементы наследуют стили родительских элементов (не все стили наследуются)

```
p { color: red; }
```

```
<p>Параграф <span>спан</span></p>
```

Каскадность

Обработка таблиц стилей по их приоритету (с наименее приоритетных):

- браузерные стили
- пользовательские стили
- внешние файлы стилей (style.css)
- стили внутри html (`<style></style>`)
- стили в атрибуте тэга (`style="color: green;"`)
- `{ color: red !important; }`

Важен порядок описания стилей внутри одной таблицы стилей

Итого:

На конечные свойства элемента влияет:

- Как подключаются таблицы стилей (каскад)
- В каком порядке описаны стили внутри одной таблицы стилей (каскад)
- Какие свойства наследуются от родителей (наследование)
- Какие селекторы используются (специфичность)

Base properties

Оформление текста

```
color: color;

font: font-size/line-height font-weight font-style
font-family;

text-decoration: underline | none | …;

text-transform: uppercase | none | …;

text-align: center | justify | left | right | …;

text-shadow: <x> <y> <размытие> <цвет>;

vertical-align: baseline | bottom | middle | top | …;
```

Бордеры, бэкграунды, градиенты, тени

border: border-width border-style border-color;
border-top/right/bottom/left;

border-radius: <px> | <%>;

background: background-image (linear-gradient | radial-gradient) background-repeat background-position background-color; (несколько изображений)

background-size: <px> | <%> | auto | cover | contain;

box-shadow: inset <x> <y> <размытие> <растяжение> <цвет> (несколько теней)

Другие свойства

```
visibility: visible | hidden;

opacity: [0.0 - 1.0];

cursor: pointer | auto | text | …;

outline: outline-color outline-style outline-width;

overflow: auto | hidden | scroll | visible;

white-space: normal | nowrap | …;

columns: column-width column-count;

appearance: none | button | field | …;

all: initial | inherit | unset | revert;
```

Units

Единицы измерения

%, px, vw, vh, vmin, vmax, em, rem,
--variables, calc()

```
:root {
--light: 20%;
}
span {
  background: hsl(calc(360 / 3), 20%, var(--light, 50%));
}
```

Цвет

Hexadecimal #RRGGBB,

```
color: #00ff00; /* green */
```

rgb(red, green, blue), rgba(red, green, blue, alpha):

```
color: rgba(0, 255, 0, .5);}   /* green 0.5 opacity */
```

hsl(hue, saturation, lightness), hsla(hue, saturation, lightness, alpha):

```
color: hsl(120, 100%, 50%);}   /* green */
```

Предустановленные цвета:

```
color: green;   /* green */
```

Директивы CSS (At-Rules)

```
@import "/style/main.css";

@font-face { … }

@media all and (orientation: landscape), all and (min-width: 480px) { … }

@supports (display: flex) and (-webkit-appearance: checkbox) { … }

@keyframes mymove { from { … } to { … } }
```

Pseudo elements

Псевдо-элементы

```
::before {} ::after {}


{
  content: строка | attr(параметр) | counter;
}
```

Контексты форматирования

Контексты форматирования

Контексты форматирования какие бывают?

`display:`

Контексты форматирования какие бывают?

```
display:
(none, contents)

inline, inline-block,
block, flow-root,
table, inline-table, (table-row, table-cell, ...)
flex, inline-flex,
grid, inline-grid,

list-item
ruby, (ruby-base, ruby-text, ...)
```

Контекст форматирования внешний

Инлайновый контекст (встраиваются в текст, сохраняют пробелы в коде вокруг себя, подчиняются свойству vertical-align)

Блочный контекст (идут по вертикали, их вертикальные margin-ы схлопываются с соседями, окружение становится блочным: разбивают текст на анонимные блоки)

Другое: на соседей не влияет (none), либо поведение зависит от внешнего контекста (table-*).

Контекст форматирования внутренний

Создают свой новый внутренний контекст форматирования:

```
table, table-*
flex
grid
inline-block (новый нормальный поточный контекст)
flow-root (новый нормальный поточный контекст)
```

Продолжают родительский нормальный поток (не создают новый):

```
inline
block
```

Блочный контекст

Блочная модель

- Занимают доступную ширину родителя.
- Блоки располагаются по вертикали друг под другом.
- Маргины схлопываются у соседних и у вложенных элементов (в нормальном потоке документа).

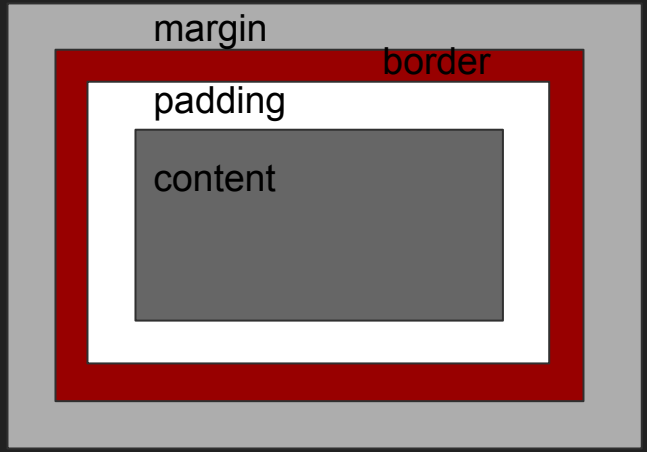
`width: значение | % (от ширины родителя) | auto | min-content | max-content | fit-content`

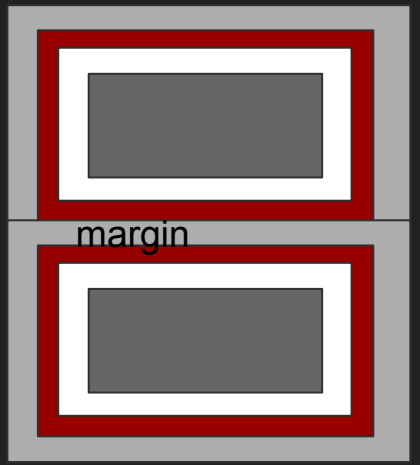
`height` (в % от высоты родителя, если она задана явно)

`margin` (в % от ширины родителя), `auto`

`padding` (в % от ширины родителя)

`box-sizing: content-box` (по дефолту) | `border-box` | `padding-box`





Float

Float

float

- Элемент отображается как блочный, так словно ему установлено свойство display: block;
- Элемент по ширине сжимается до размеров содержимого, если для элемента явно не установлена ширина width;
- Элемент прилипает к левому (left) или правому краю (right);
- Плавающие блоки не влияют на обычные, но влияют на их инлайновое (текстовое) содержимое (содержимое страницы, идущее в HTML коде после элемента с float, обтекает его);

float блоки разной высоты

Свой контекст форматирования для float-ов

`clear: both;` (не создает контекст форматирования, а очищает float)

создают свой контекст форматирования:

```
overflow: hidden | auto;
display: inline-block;
display: flow-root;
```

…

Position

Position

`position: static` (default) `| relative | fixed | absolute |` `sticky`

`left, right, bottom, top`

Выравнивание блока absolute по центру

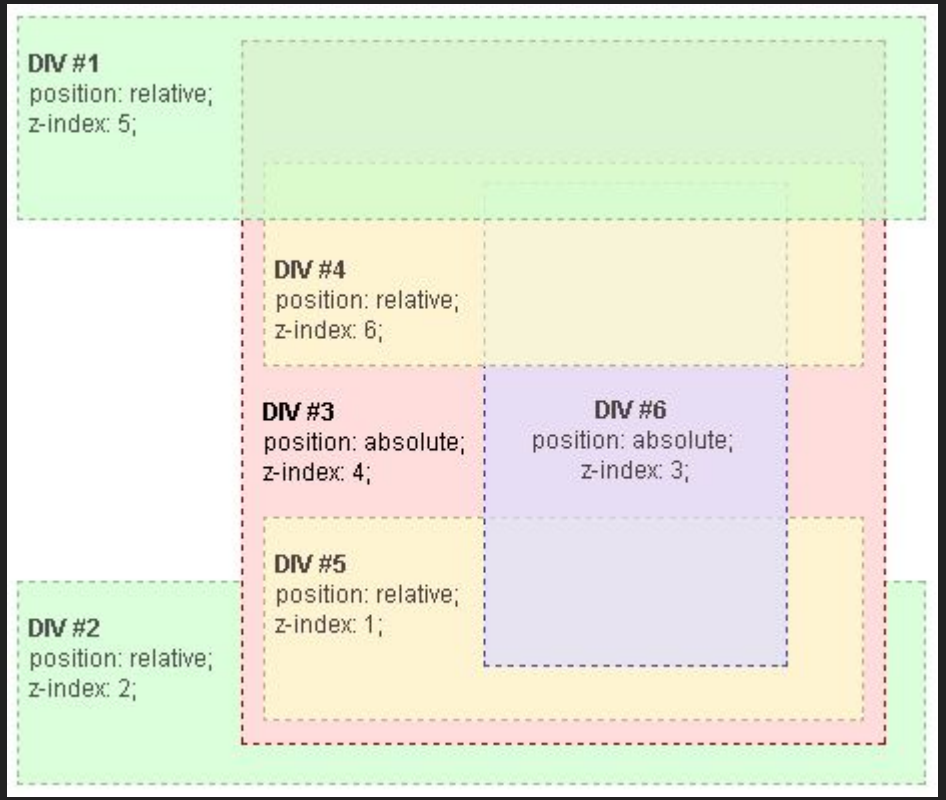
Контексты наложения (stacking context)

Z-index

`Position (relative, absolute, fixed, sticky)` создает контекст наложения, когда `z-index` не “auto”.

`Z-index` указывает позицию по Z.

```
DIV #1
DIV #2
DIV #3
        DIV #4
        DIV #5
        DIV #6
```



Инлайновый контекст

Строчные элементы

Непосредственная часть строки, элементы идут друг за другом.

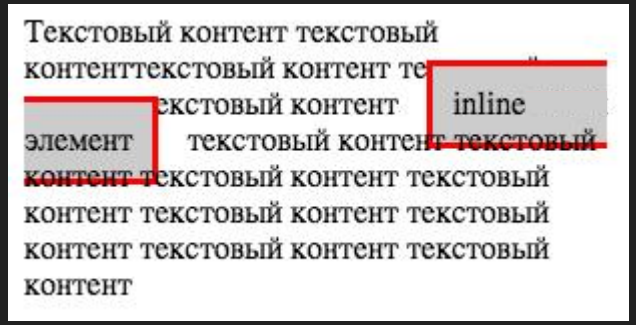
Можно выравнивать по вертикали с помощью **vertical-align**.

Перенос текста считается за пробел.

display: inline

Не работают width, height и вертикальные отступы.

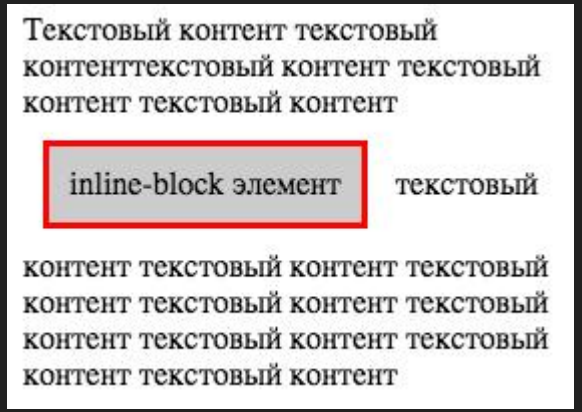
Переносятся на другую строку по словам.



display: inline-block

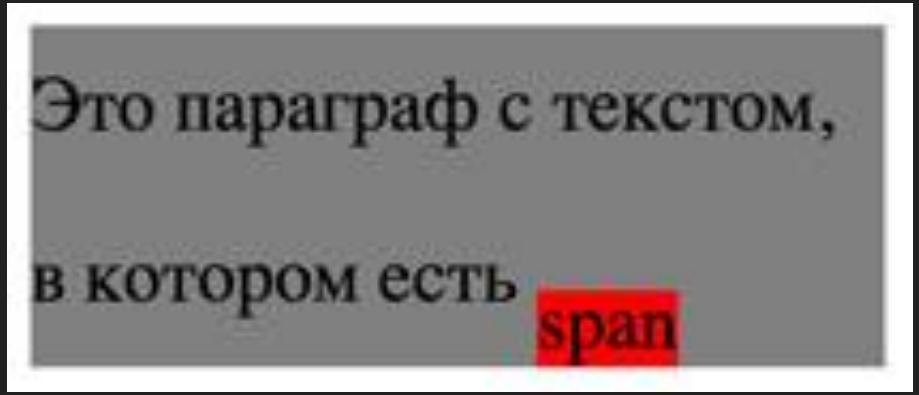
Можно устанавливать width и height.

Переносятся на другую строку целиком.

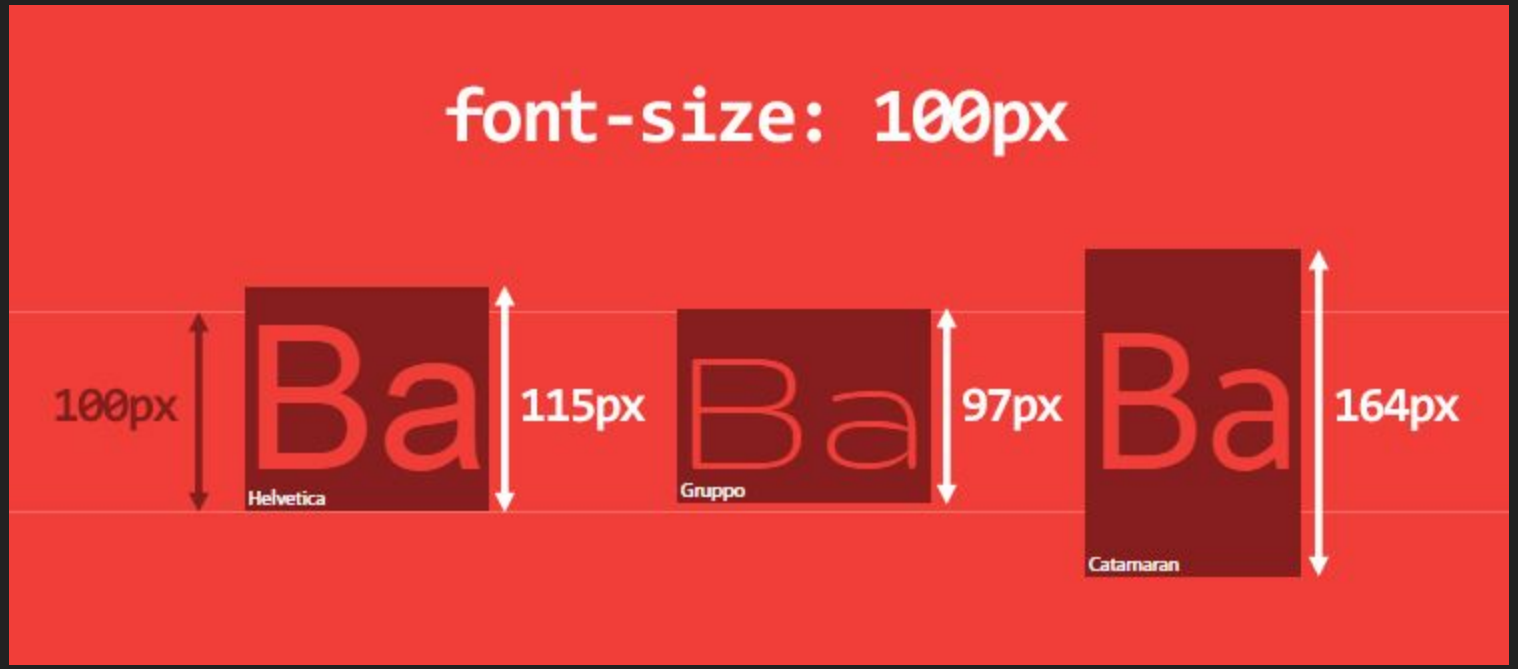


Метрики шрифта, line-height и vertical-align

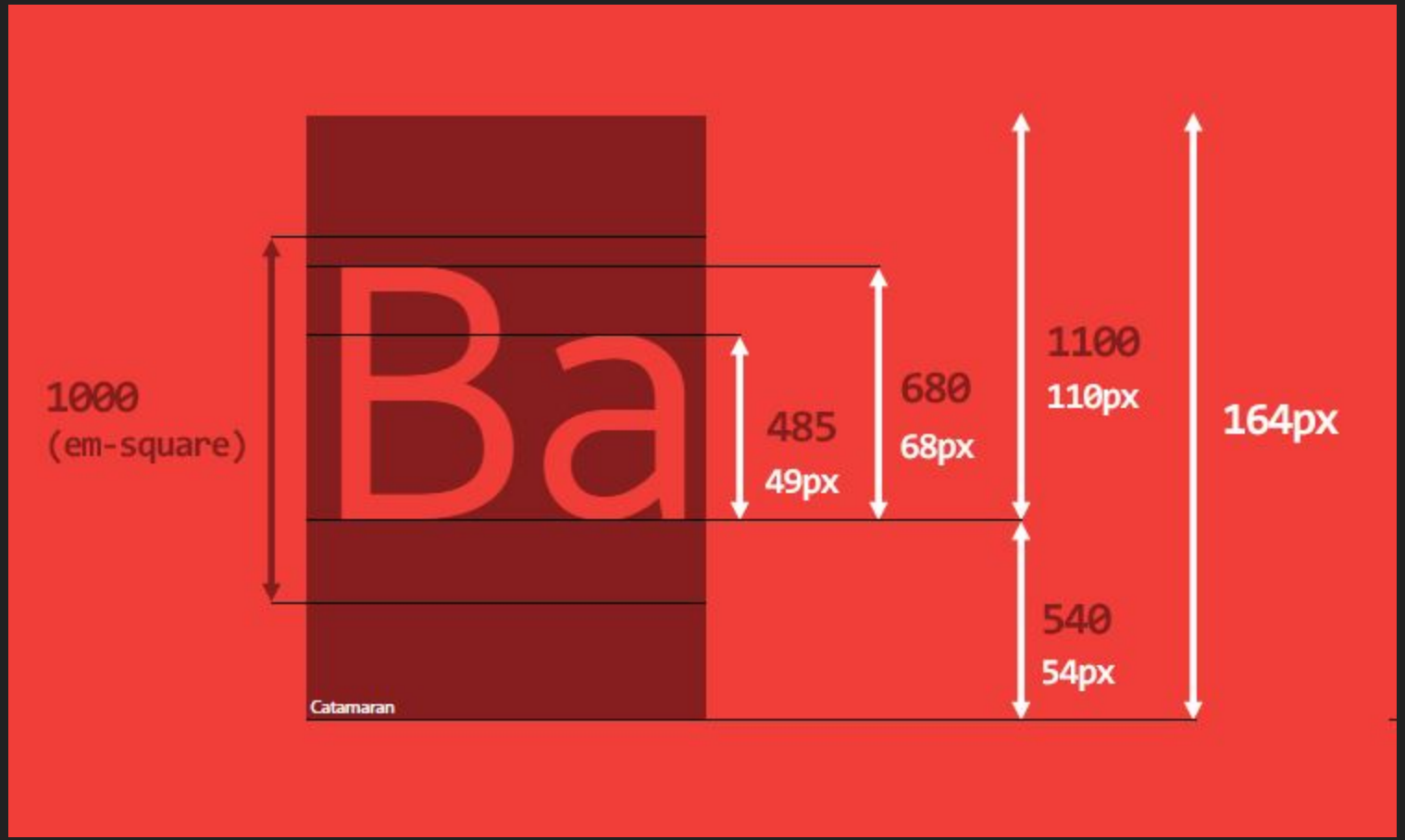
Как работает выравнивание текста?



Сколько по высоте будет font-size: 100px?



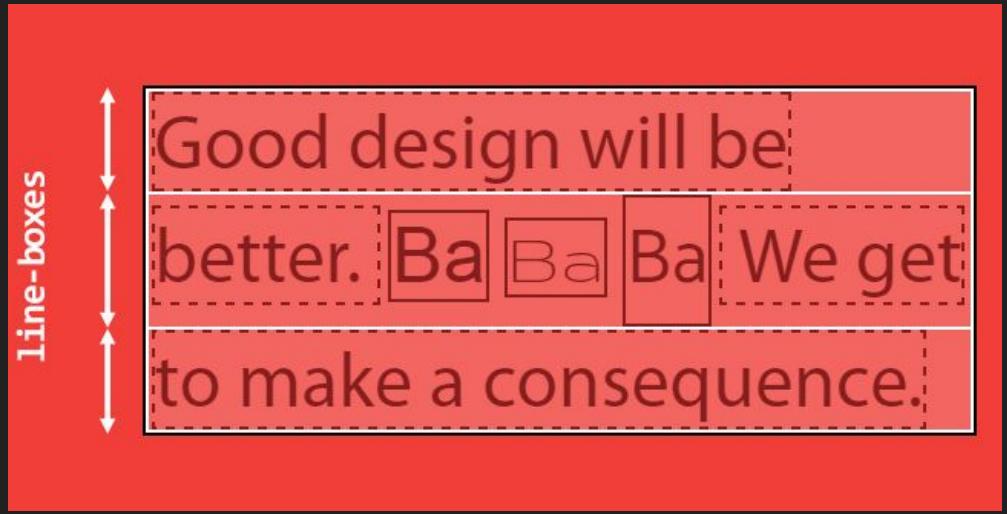
Скрины из статьи Vincent De Oliveira “Deep dive CSS: font metrics, line-height and vertical-align”


Ba
1000
(em-square)
485
49px
680
68px
1100
110px
164px
540
54px
Catamaran

Как определяется Y-координата каждой новой строки текста?

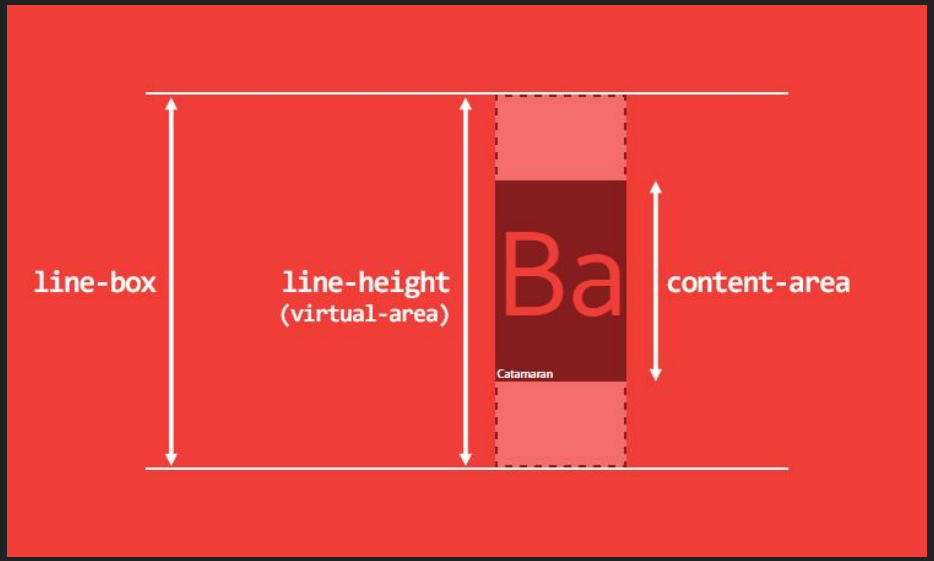
Виртуальные строки (line-box)

```
<p>
    Good design will be better.
    <span class="a">Ba</span>
    <span class="b">Ba</span>
    <span class="c">Ba</span>
    We get to make a consequence.
</p>
```

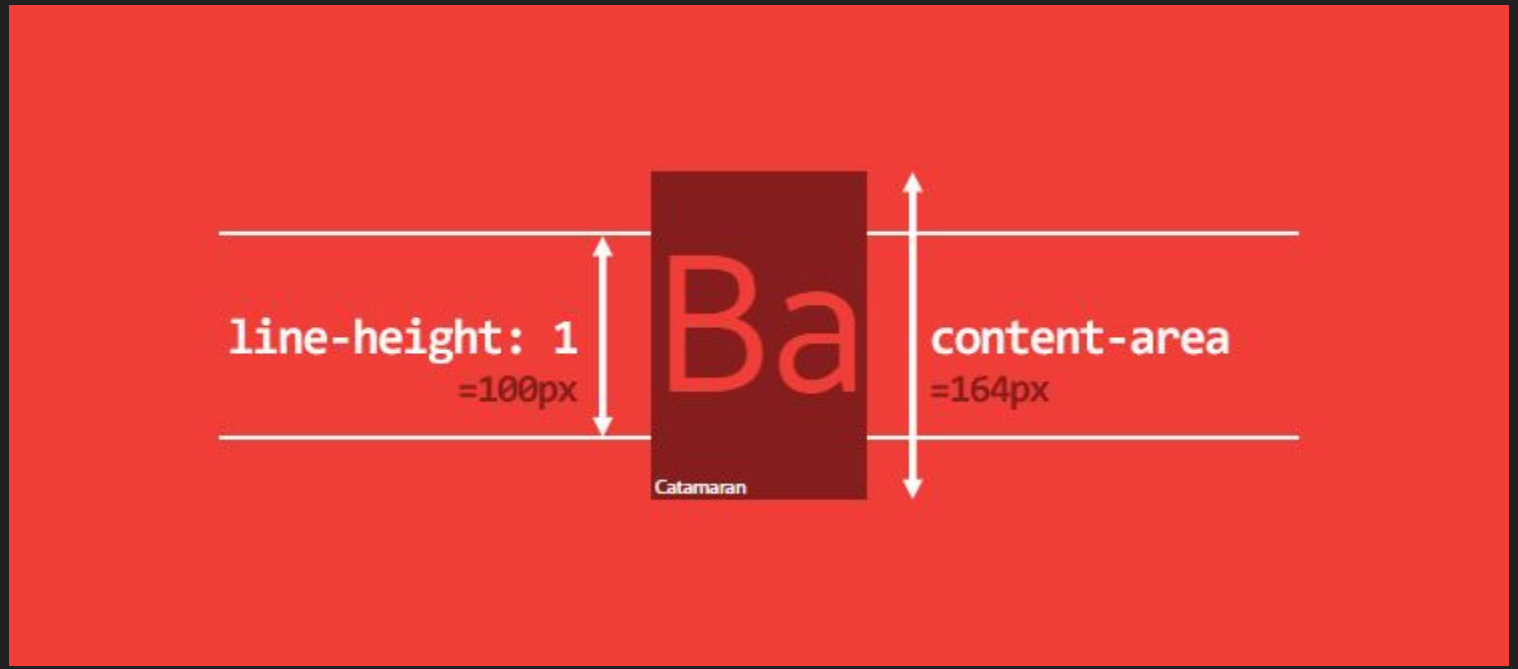


Высота виртуальной строки

Зависит от области содержимого и line-height.
line-height: normal задается отдельно каждому шрифту в его параметрах.
line-height в относительных единицах считается от размера шрифта в CSS.

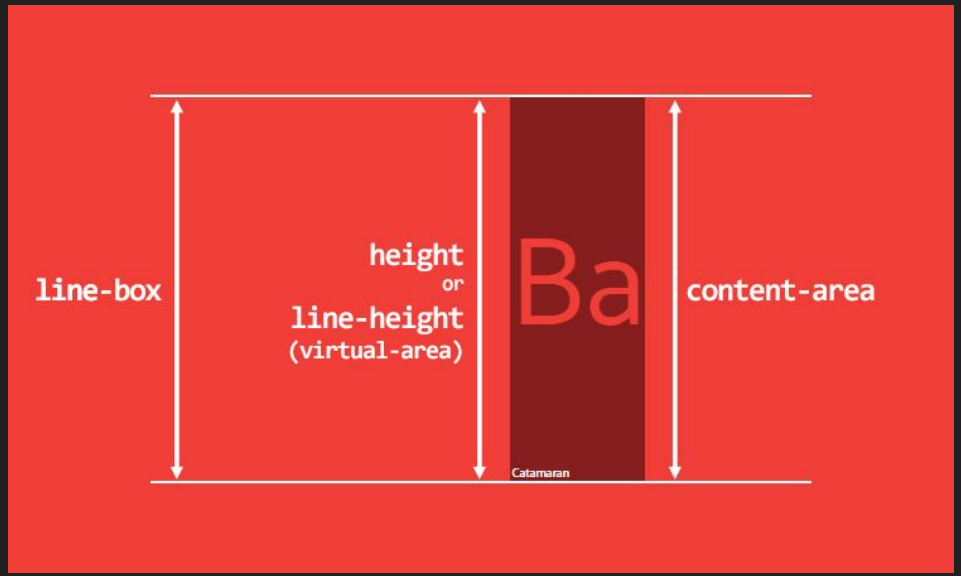


Из-за line-height: 1 контейнер строки может стать ниже, чем область содержимого



Высота строки может быть равна height

У inline-block/inline-*, если высота установлена явно.

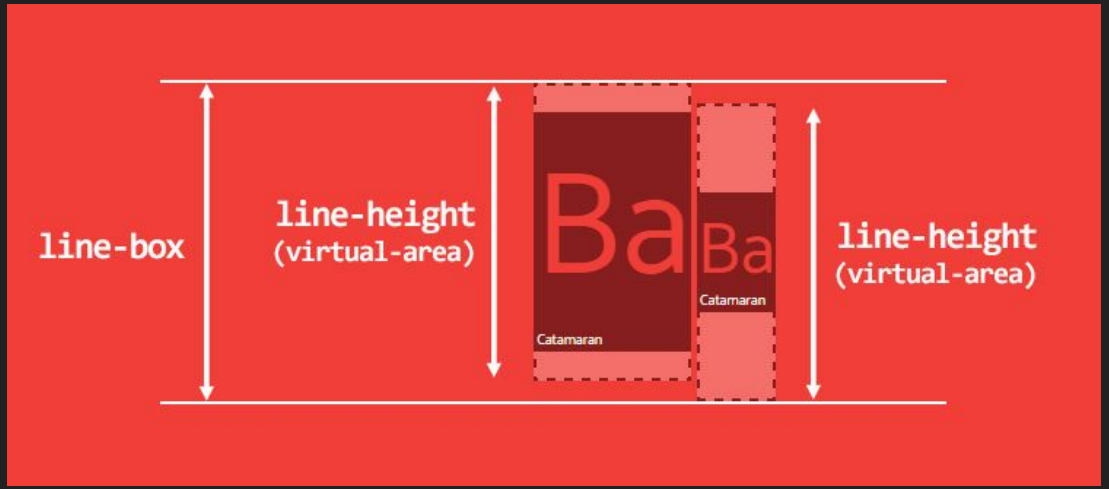


Что еще влияет на высоту контейнера строки

Высота контейнера строки рассчитывается от самой высокой до самой низкой точки его потомков

Vertical-align (baseline по умолчанию)

```
<p>
    <span>Ba</span>
    <span>Ba</span>
</p>
p {
    font-family: Catamaran;
    font-size: 100px;
    line-height: 200px;
}
span:last-child {
    font-size: 50px;
}
```

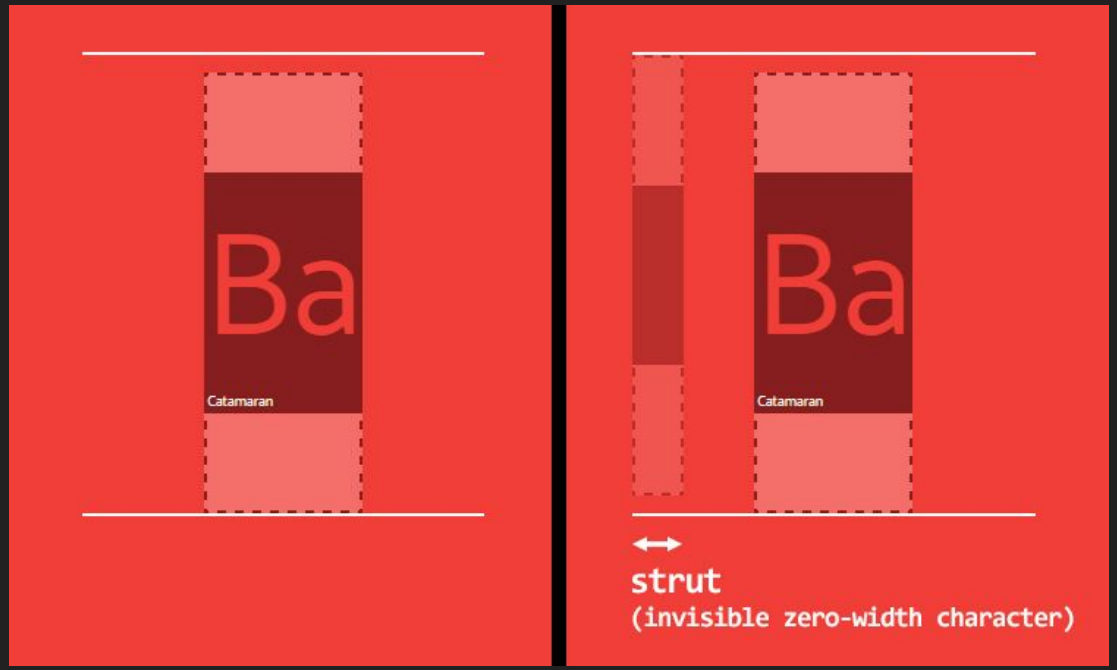


Какой высоты будет контейнер строки?

```
<p>
    <span>Ba</span>
</p>
p {
    line-height: 200px;
}
span {
    font-family: Catamaran;
    font-size: 100px;
}
```

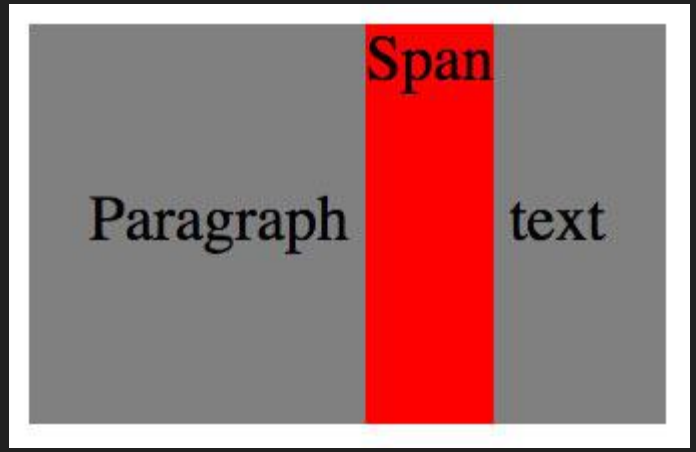
Какой высоты будет контейнер строки?

```
<p>
    <span>Ba</span>
</p>
p {
    line-height: 200px;
}
span {
    font-family: Catamaran;
    font-size: 100px;
}
```

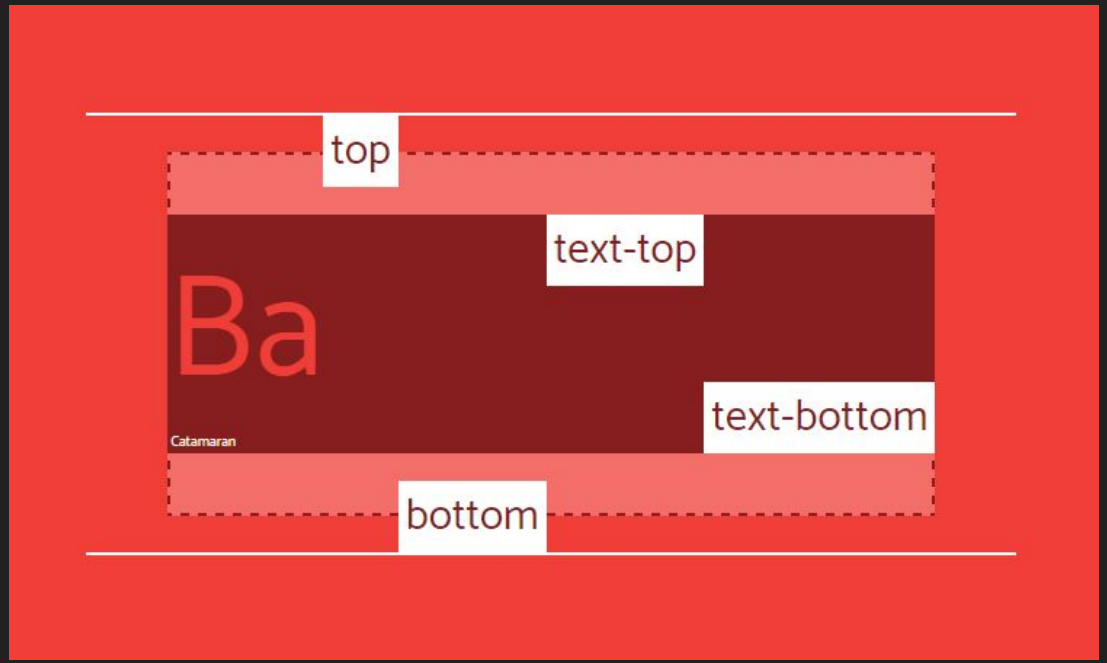


vertical-align: middle тоже привязан к базовой линии

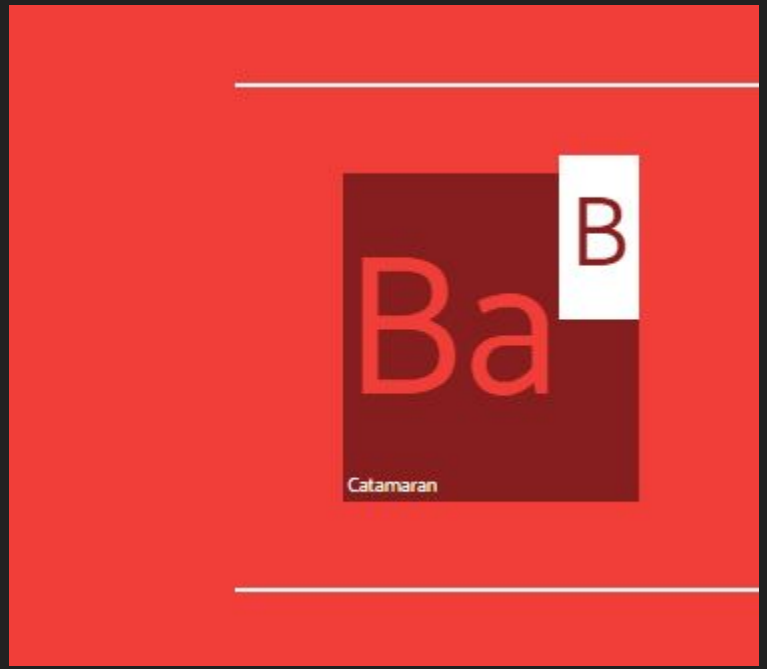
```
<p>Paragraph
  <span>Span</span>
  text
</p>


p {
  background: gray;
}
span {
  background: red;
  display: inline-block;
  height: 100px;
  vertical-align: middle;
}
```



Другие значения vertical-align (отвязаны от базовой линии)

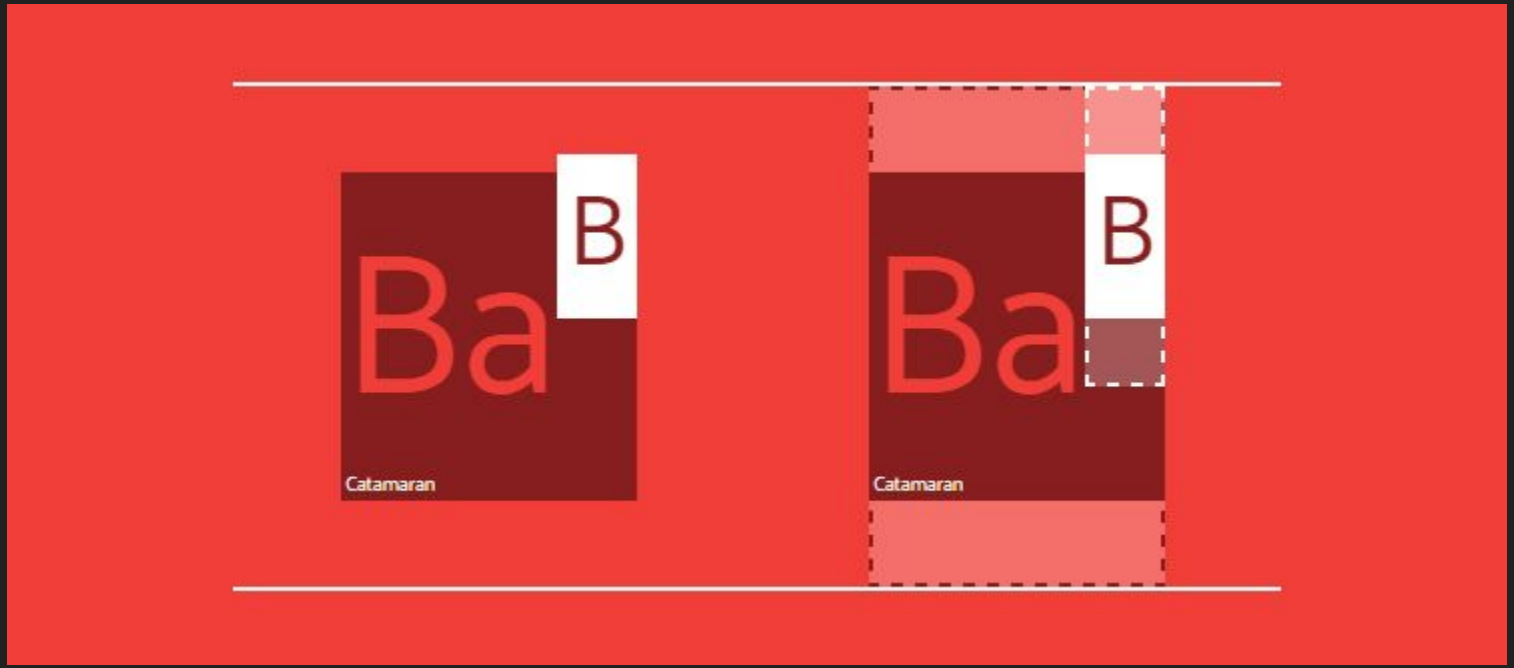


vertical-align выравнивает виртуальную область (line-height), высота которой невидима:



vertical-align: top;

vertical-align выравнивает виртуальную область (line-height), высота которой невидима:



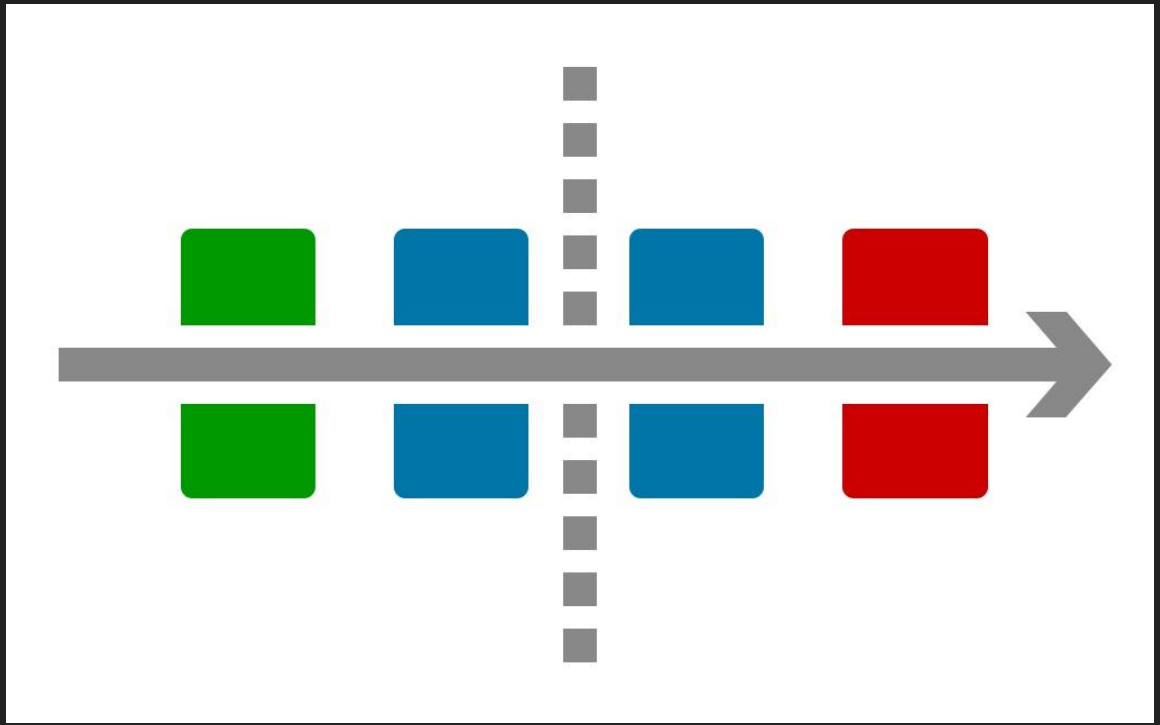
Итого

- у всех строчных элементов 2 высоты: область содержимого (основанная на метриках шрифта) и виртуальная область (line-height)
- font-size зависит от соотношения 1 em-квадрата к другим метрикам шрифта
- line-height: normal основывается на метриках шрифта
- line-height: n основывается на font-size шрифта, и при его использовании виртуальная область может стать меньше, чем область содержимого
- высота контейнера строки рассчитывается на основе свойств line-height и vertical-align его потомков

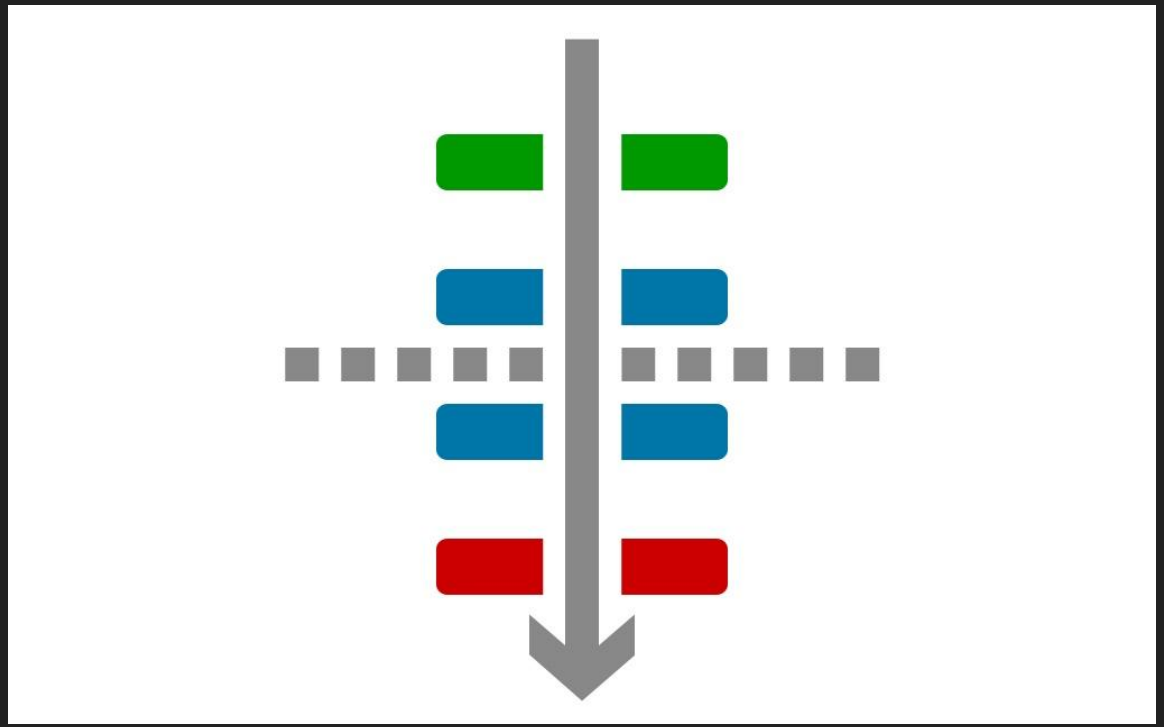
Выравнивание

Flexbox

Flexbox



Flexbox



Flexbox

```
display: flex | inline-flex;
flex-direction: row | row-reverse | column | column-reverse;
flex-wrap: nowrap | wrap | wrap-reverse;
flex-flow: row nowrap (краткая запись flex-direction и flex-wrap)
order: number;
flex-grow: number;
flex-shrink: number;
flex-basis: value;
flex: 0 1 auto; (краткая запись flex-grow flex-shrink flex-basis)
```

Flexbox (выравнивание)

`justify-content: flex-start | flex-end | center | space-between | space-around;`

`align-items: stretch | flex-start | flex-end | center | baseline;`

`align-self` (для одного элемента переопределяет выравнивание, заданное align-items)

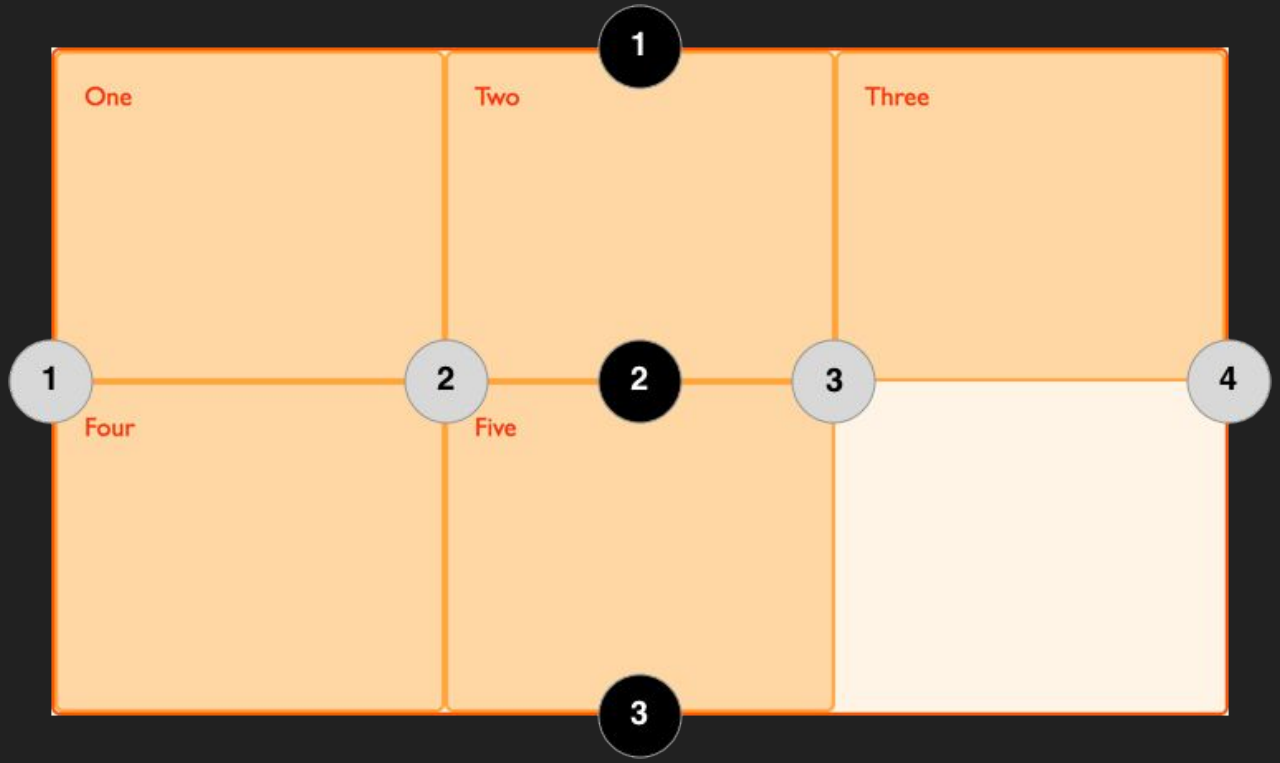
`align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch;` (для многострочного флекса, и если место позволяет)

Выравнивание flexbox-айтема по центру

https://flexboxfroggy.com

Grid layout

Grid layout



Grid layout

```
display: grid | inline-grid;

grid-template-columns, grid-template-rows: fr | minmax(10px, auto)| repeat();

grid-column-gap, grid-row-gap: value;

grid-row-start, grid-row-end, grid-column-start, grid-column-end: number of track | [span]

grid-auto-rows, grid-auto-columns: value;

grid-auto-flow: row | column | dense;
```

Grid layout (выравнивание)

justify-items (justify-self): stretch | start | end | center;

align-items (align-self): stretch | start | end | center;

justify-content: start (default) | end | space-between | space-around | space-evenly;

align-content: start (default) | end | space-between | space-around | space-evenly;

Grid layout

http://cssgridgarden.com/

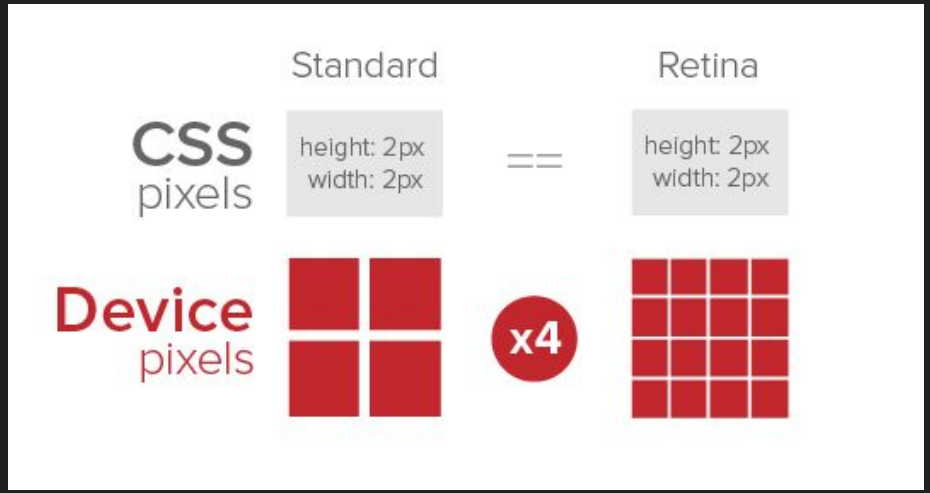
Adaptive

Adaptive

```
<meta name="viewport" content="width=device-width,
initial-scale=1">
```

```
@media all and (orientation: landscape), all and (max-width:
670px) { … }
```

Графика
(физические
и виртуальные пиксели)



Transformation

Transform

```
transform: translate | translate3d | scale | rotate[X][Y] | … ;

transform-style: flat | preserve-3d;

transform-origin: x y z;

perspective: value;
```

Выравнивание блока по центру

Animation

Transition

```
transition (transition-property, transition-duration,
transition-timing-function и transition-delay): top 1s
ease-out 0.5s;
```

Animation

```
@keyframes spin {
  from { transform: rotateX(0); }
  50%  { transform: rotateX(30deg); }
  to   { transform: rotateX(360deg); }
}
animation (animation-name animation-duration): spin 300ms;

animation-timing-function: ease | linear | … ;
animation-delay: s | ms;
animation-iteration-count: number | infinite;
animation-direction: alternate | alternate-reverse | normal | reverse;
animation-play-state: paused | running;
animation-fill-mode: none | forwards | backwards | both;
```

Производительность

Отрисовка страницы

DOM → CSSOM → Render Tree → Layout → Paint → Composite

-**Repaint** (background-color, border-color, visibility, …)

-**Reflow / Relayout** (манипуляции с DOM, расчёт/изменение CSS-свойств, добавление/удаление таблиц стилей, обращение из JS к свойствам элементов, изменения размеров, прокрутка окна, hover, …)

Оптимизация

- Анимировать желательно только абсолютно и фиксировано спозиционированные элементы
- В скриптах минимизируйте любую работу с DOM
- Для изменения стилей элементов лучше модифицировать только атрибут «class», и как можно глубже в дереве DOM
- Выносить анимируемые элементы в композитные слои (3D-трансформация, CSS анимации opacity/transform, CSS фильтры, <canvas>, ...)
- Выносить композитный элемент над другими (z-index), тогда он не будет влиять на другие слои

Filters

Filters, Mask, Clip-path, …

filter: blur | drop-shadow | grayscale, …;

shape-outside: polygon(0 0, 100% 0, 100% 10%, 0 100%);

clip-path: polygon(0 0, 100% 0, 100% 10%, 0 100%);

mask: url(mask.png);

mix-blend-mode: difference | color | luminosity | color-dodge | …;

Препроцессоры

scss

переменные, миксины, экстенды, вложенность, циклы, функции, списки, массивы, импорт файлов

https://www.sassmeister.com/

Вопросы