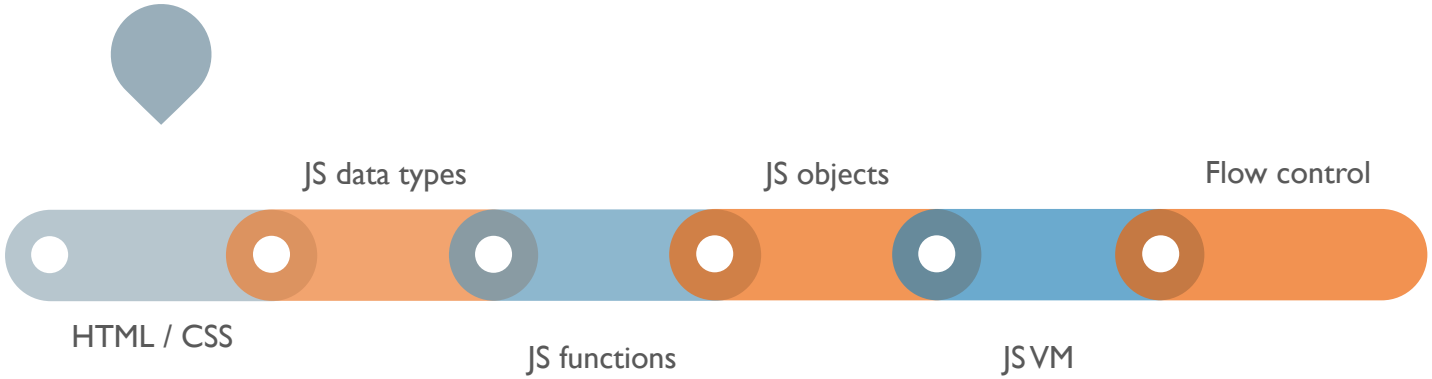
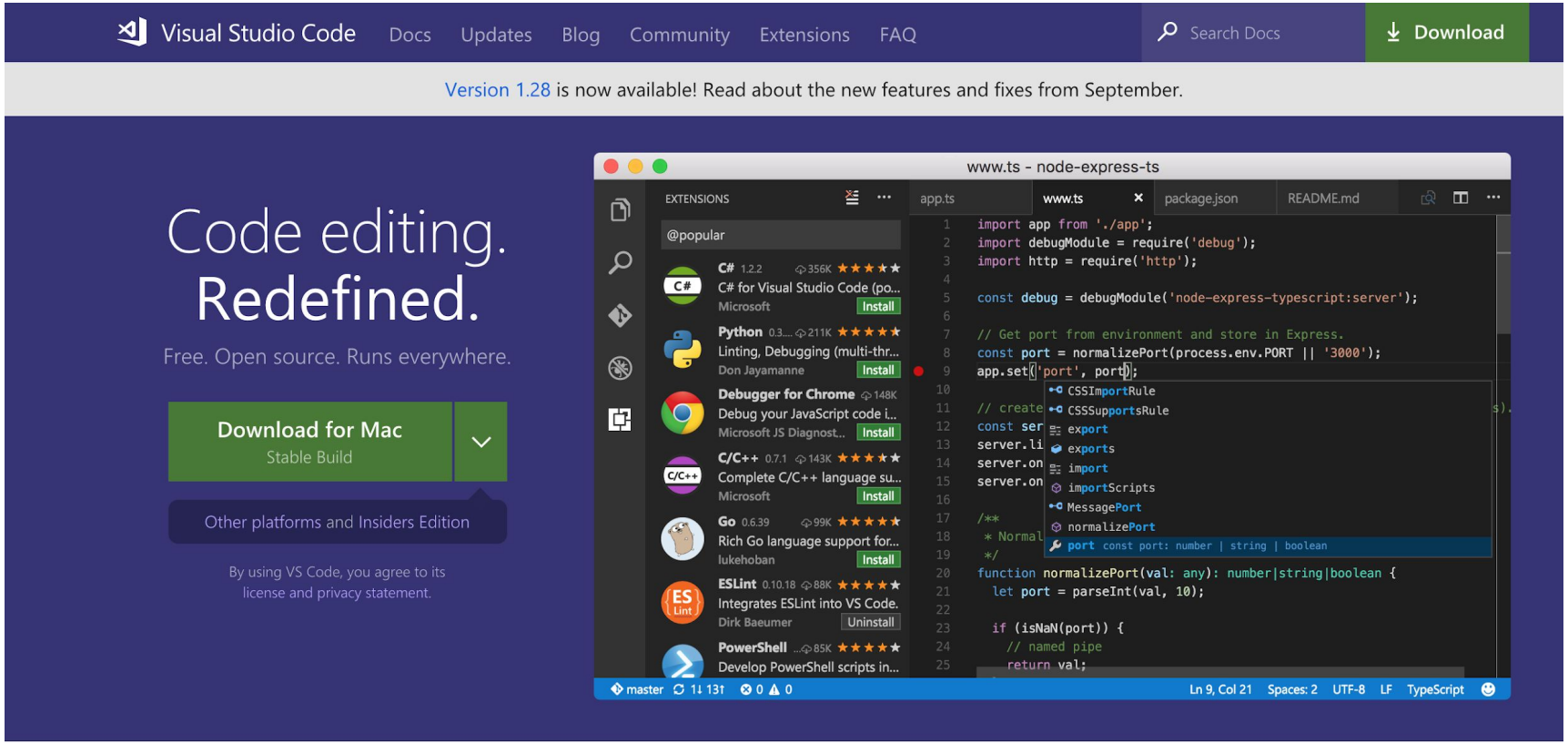
Лекция 1


JS data types
JS objects
Flow control
HTML / CSS
JS functions
JS VM

Set up IDE

Visual Studio Code

http://code.visualstudio.com



IntelliSense

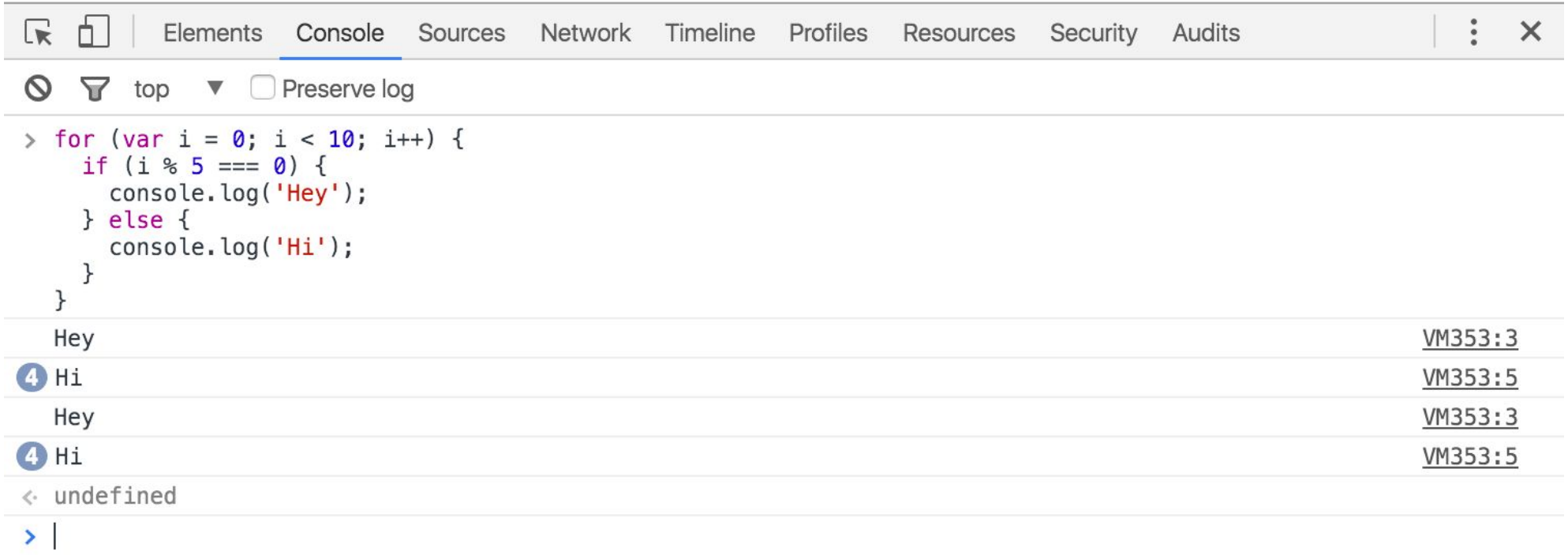
Debugging

Built-in Git

Extensions

Chrome console

- On Mac - ⌘ **+ Shift + C**.
- On Windows / Linux - **Ctrl + Shift + C** or **F12**.



Hello world

```
<html>
   <head>
      <title>
         Simple Web Page
      </title>
   </head>
   <body>
      <h1>Hello World!</h1>
      <a href="#">Link</a>
   </body>
</html>
```

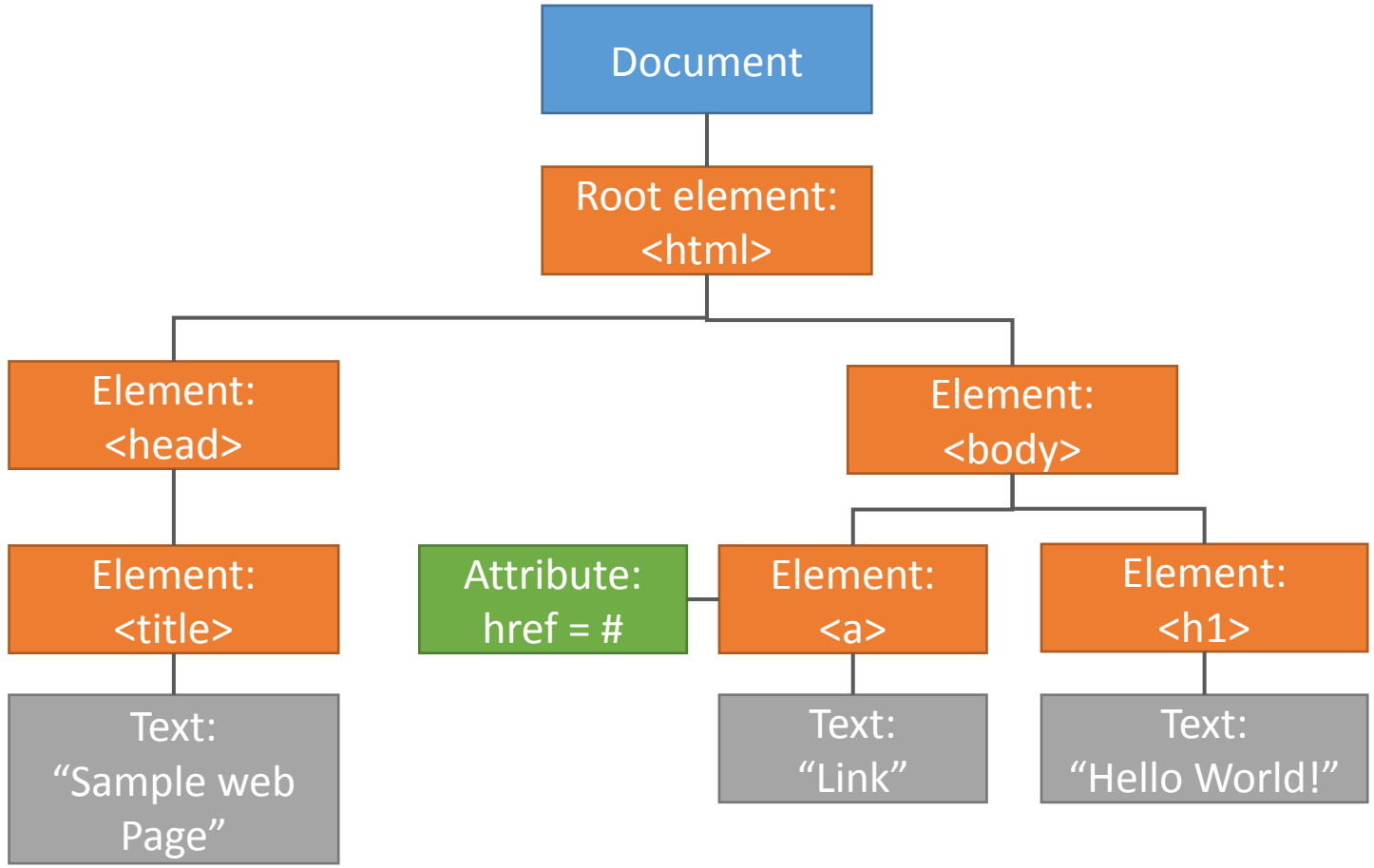
HTML

HTML

- **HTML** страниците са **текстови документи**
- HTML е изграден от **елементи**, наречени **тагове**
- Таговете могат да имат **атрибути**, които да имат присвоени **стойности**
- Един HTML документ се представя като дърво от елементи (обекти) и формира **D**ocument **O**bject **M**odel (**DOM**)
- **DOM** дефинира **елементите**, **свойствата** на елементите, **методи за достъп** до елементите и **събития** (events), които се пораждат от елементите

HTML **D**ocument **O**bject **M**odel (DOM)

```
<html>
   <head>
      <title>
         Simple Web Page
      </title>
   </head>
   <body>
      <h1 id=”neshto”>Hello World!</h1>
      <a href="#">Link</a>
   </body>
</html>
```



DOM manipulations

HTML DOM предоставя методи, които могат да манипулират структурата и елементите на модела.
Често тези функции са обвити в библиотеки като jQuery, MooTools, Umbrella и други.
Пример за такива методи:

- Достъп до елемент по атрибут **идентификатор** (име): document.getElementById(”id_name”);
- Списък от елементи **по име на таг**: document.getElementsByTagName(”tag_name”);
- Списък от елементи по атрибут **име на клас**: document.getElementsByClassName(”class names”);

Тези методи връщат като резултат единична инстанция или списък от обекти от тип Node (елемент в DOM).
Той също предоставя методи за манипулация на атрибутите си и елементите наследници (деца).
Пример за такива методи:

- Списък на всички деца на елемент: node.childNodes();
- Добавяне на наследник на елемент: node.appendChild(element_object);
- Достъп до стойност на атрибут на елемента: node.getAttribute(”attribute_name”);
- Създаване на **нов елемент** в DOM: document.createElement(”tag_name”);

DOM Events

DOM събитията позволяват да регистрираме функция към определен елемент, която се извиква тогава,
когато се случи нещо с елемента. Например натискане на бутон на мишката върху този елемент.
Регистрирането на функция към събитие може да стане по няколко начина.

Като атрибут в HTML тага на елемента:

```
<button id="theButton" onclick="someFunction()">Click me</button>
```

Използвайки event метода:

```
document.getElementById("theButton").onclick = function() { console.log('Clicked'); };
```

Чрез функцията за регистрация към събитие:

```
document.getElementById("theButton").addEventListener("click", someFunction);
```

CSS

Cascading Style Sheets (CSS)

CSS ни дава набор от правила, чрез които да задаваме как ще изглежда и къде ще се появява съдържанието на всеки елемент от страницата.

Чрез създаването на стилове можем да управляваме визуализацията на елементите.

Стиловете представляват съвкупност от параметри и техните стойности, които ще се вземат предвид

по време на изчертаване на елемента върху екрана.

Пример за такива параметри са:

- Цвят на фона на елемента.
- Шрифт на текста на елемента.
- Позиция на елемента в документа или спрямо другите елементи.

CSS Selectors

CSS позволява асоцииране на правила, които описват визуален стил към дадени HTML елементи.
Всяко правило се състои от две части: **селектор** и **декларация**.
Селекторът определя към кои елементи или елемент ще се асоциира стила.
Декларацията представлява описанието на визуалния стил, който ще се асоциира с елемента.

Selector **Declaration**

```
p {
    font-family: Arial;
}
```

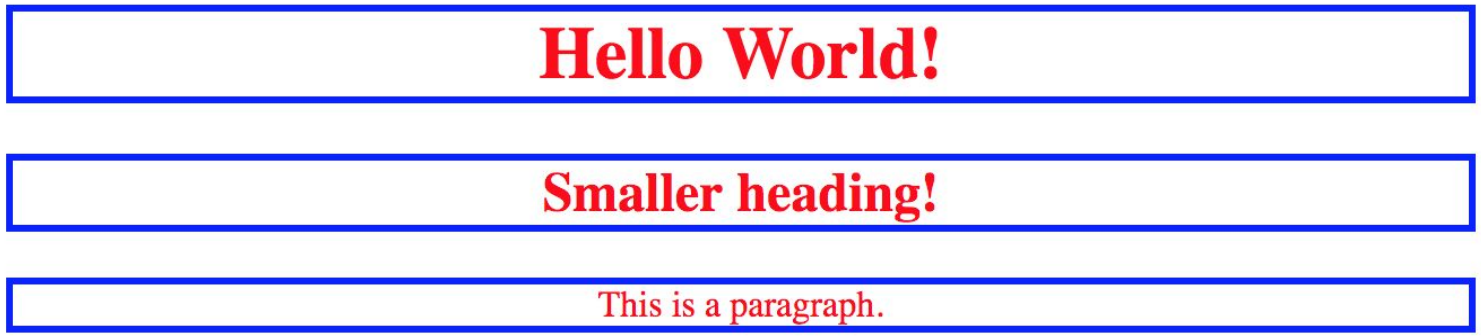
Селекторите могат да бъдат по тип на елемента, по идентификатор, по клас и други.
Те също могат и да се групират.

CSS Selectors

```
<html>
<head>
   <style>
      h1, h2, p {
         border-style: solid;
         border-width: 1;
         border-color: blue;
         text-align: center;
         color: red;
      }
   </style>
</head>
<body>

<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
<p>This is a paragraph.</p>

</body>
</html>
```



CSS box model

За CSS, всеки елемент представлява правоъгълник със следните свойства:

- **Margin**: отстояние от другите елементи.
- **Border**: граница на елемента,

  може да и се задава цвят и дебелина.
- **Padding**: отстояние на съдържанието на

  елемента

  от границата му.

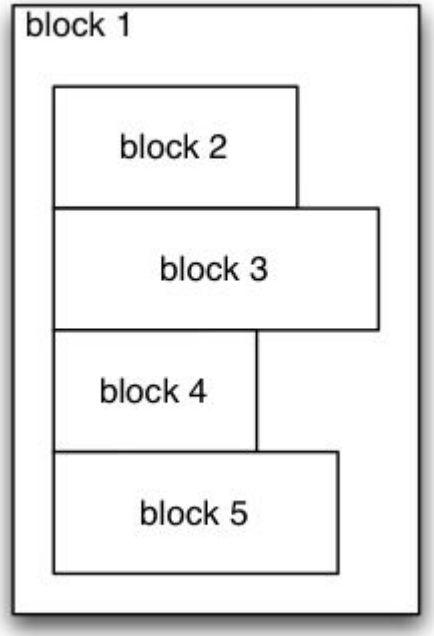
След определяне на визуалните характеристики на даден елемент следващата задача на CSS е да укаже
как и къде елементите ще бъдат разположени върху документа.

CSS Layout

CSS използва два подхода за позициониране на елементите
върху документа: **block** и **inline**.

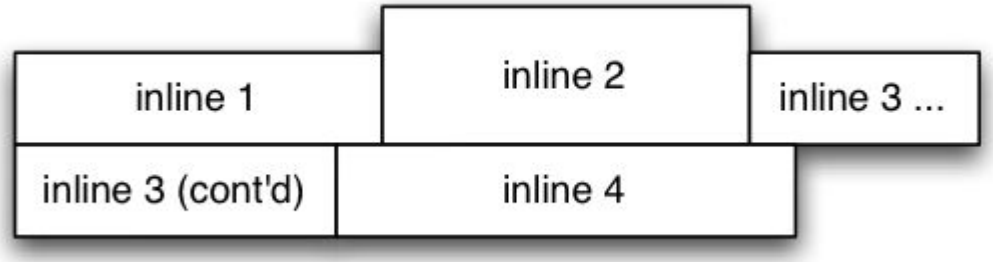
```
{ display: block; }
```

Block елементите се подреждат едни под други върху документа. Винаги започват с нов ред и заемат пространството
по цялата дължина да документа.
Те могат да съдържат други block елементи.



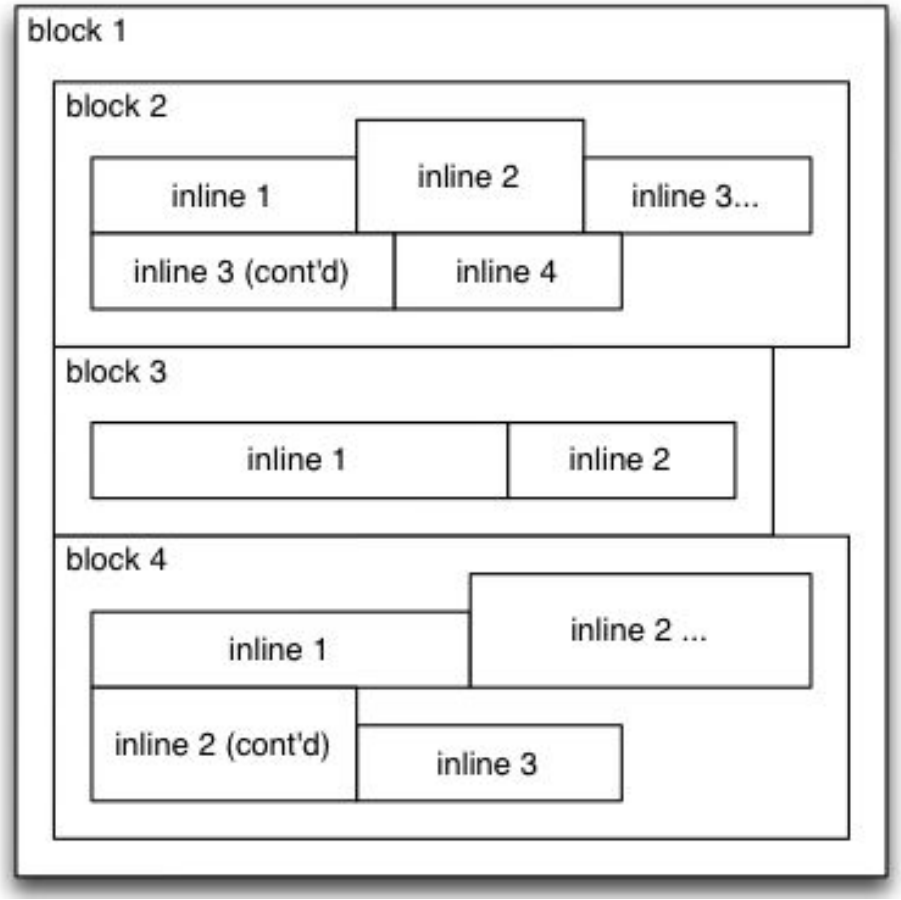

```
{ display: inline; }
```

Inline елементите не започват с нов ред и не заемат цялата дължина на документа. Вместо това те се долепят
до правоъгълника на предишния елемент и заемат пространството ограничено само от собствения им правоъгълник. Обикновено те се влагат в други елементи.



CSS Layout

За да се постигне желаното форматиране на документа се използва комбинация от block и inline елементи.

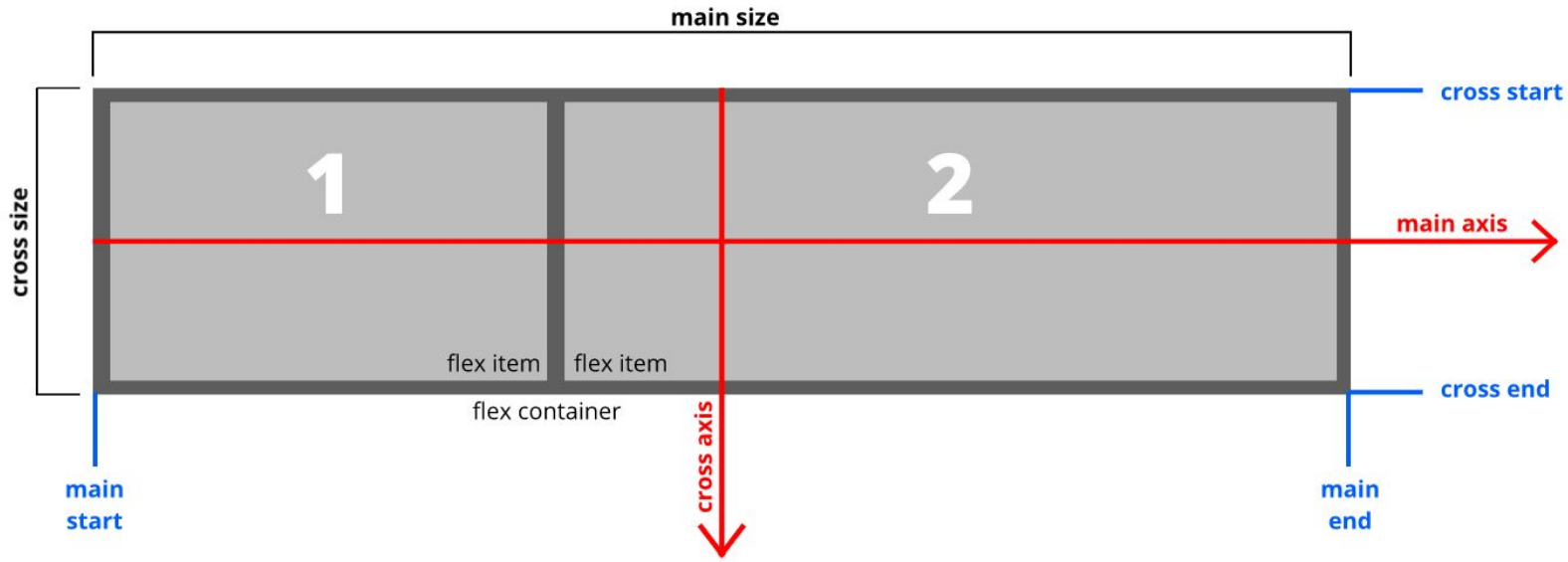


CSS Flexbox

Flexbox layout е алтернатива на класическата подредба в CSS. Целта му е да се предостави по-ефективен
начин за разпределяне на елементите по страницата без размерите на елементите да е предварително
известен.
Тук елементите са поставени в контейнер, който контролира размера им и разпределя свободното място около тях. Друга много важна особеност на контейнера е, че определя посоката в която елементите ще се разпределят без предварително да фиксира позицията им.
Това позволява да се проектират страници с динамични размери и за различни размери екрани по-лесно от колкото при класическия layout подход.

CSS Flexbox

Основни понятия при Flexbox.

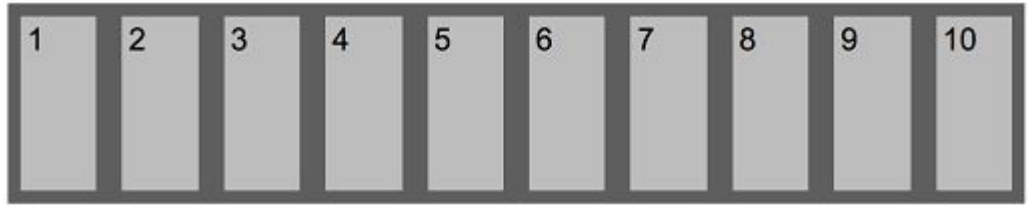


flex-direction

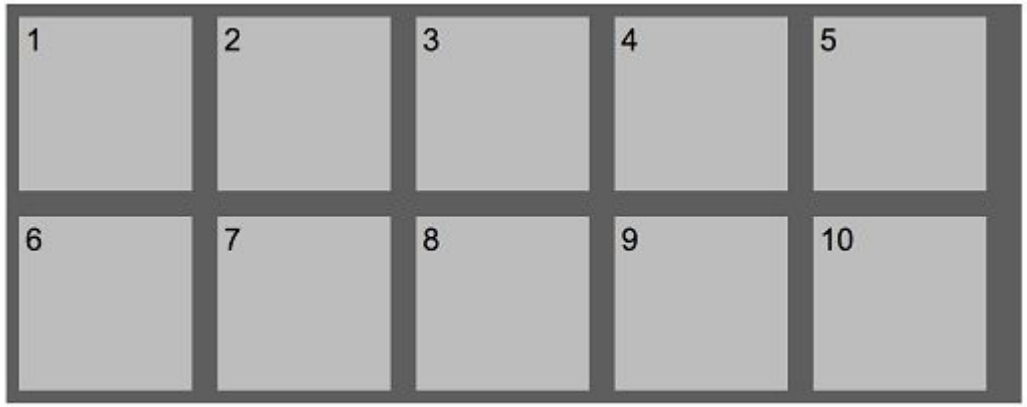
flex-direction: row; (default)

flex-direction: column;

flex-wrap

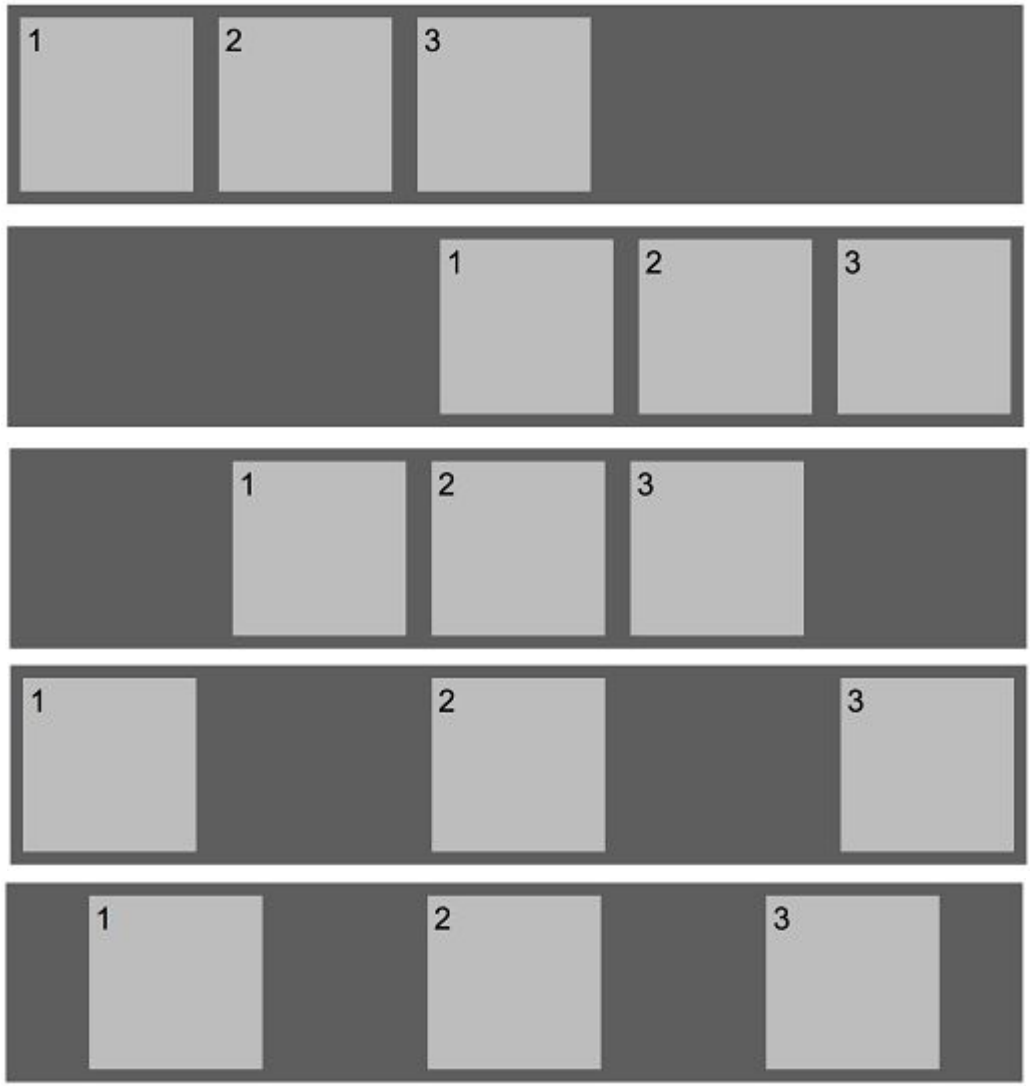


flex-wrap: nowrap;



flex-wrap: wrap;

justify-content



justify-content: flex-start;

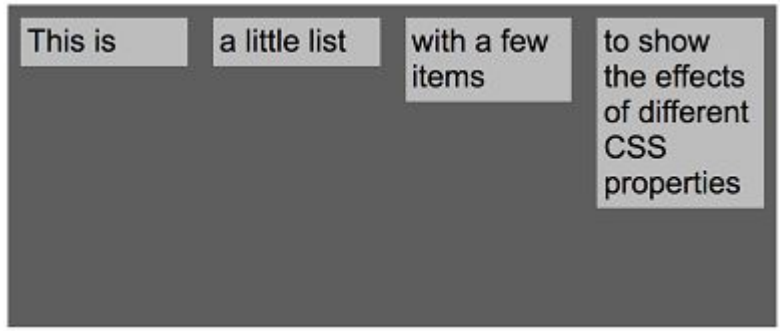
justify-content: flex-end;

justify-content: center;

justify-content: space-between;

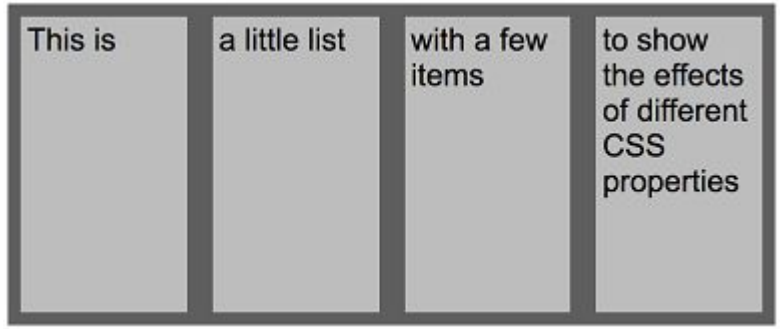
justify-content; space-around;

align-items



align-items: flex-end;

align-items: center;



align-items: stretch;

align-items: baseline;

align-content

align-content: flex-start;

align-content: flex-end;

align-content: center;

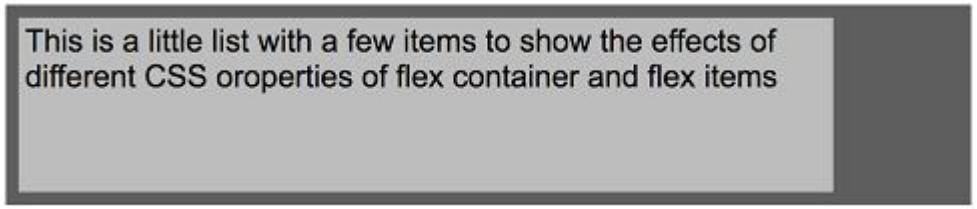
align-conent; stretch;

flex-grow

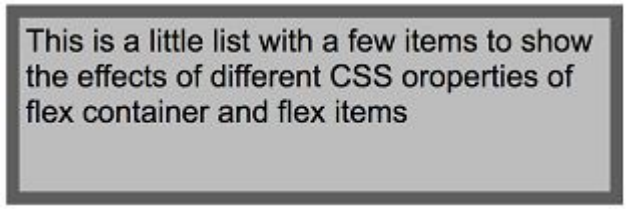
flex-grow: 0;

flex-grow: 1;

flex-shrink



width: 500px;



flex-shrink: 1;

flex-shrink: 0;

flex-basis – the initial main size of an item

flex-basis: auto;

flex-basis: auto;

flex-basis: 100px;

```
<html>
<head>
  <style>
    .flex-container {
      display: flex;
      flex-wrap: wrap;
      background-color: DodgerBlue;
    }

    .flex-container > div {
      background-color: #f1f1f1;
      width: 100px;
      margin: 10px;
      text-align: center;
      line-height: 75px;
      font-size: 30px;
    }
  </style>
</head>
<body>
<h1>Flexible Boxes</h1>
<div class="flex-container">
  <div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div>
</div>
</body>
</html>
```