

Դաս 3 - HTML տեգեր

Այս դասում մենք կուսումնասիրենք HTML տեգեր, որոնք կոգնեն մեզ ստեղծել ֆորմեր, ավելացնել պատկերներ կամ այլ ֆայլեր: Մենք կանդրադառնանք հետևյալ թեգերին՝ , <style>, <input>, <select>, <option>, <checkbox> և այլն:

 թեգը օգտագործվում է վեր էջի մեջ պատկերներ տեղադրելու համար: Այն ինքնափակվող պիտակ է, այսինքն՝ փակվող պիտակ չի պահանջում: Հիմնական հատկանիշները.

src - ուղի դեպի պատկեր:

alt - պատկերը նկարագրելու տեքստ (կարևոր է մատչելիության համար):

width և height — սահմանել պատկերի չափերը:

<style> թեգը օգտագործվում է CSS ոճերը ուղղակիորեն HTML փաստաթղթում ավելացնելու համար: Այն պետք է լինի <head> բաժնում:

```
<head>
<style>
body { background-color: #f0f0f0; }
h1 { color: blue; }
</style>
</head>
```

<input> թեգը օգտագործվում է ձևի տարրեր տարրեր ստեղծելու համար, ինչպիսիք են տեքստային դաշտերը, կոճակները և վանդակները: Հիմնական տեսակները.

type="text" - տեքստային դաշտ:

type="password" — գաղտնաբառի դաշտ:

type="checkbox" - չեկբոկս:

type="radio" — ռադիո կոճակ:

type="submit" - ֆորման ուղարկելու կոճակ:

```
<form>
  <label for="username">Имя пользователя:</label>
  <input type="text" id="username" name="username" required>

  <label for="password">Пароль:</label>
  <input type="password" id="password" name="password" required>

  <input type="submit" value="Отправить">
</form>
```

<select> և <option> թեգերն օգտագործվում են բացվող ցուցակներ ստեղծելու համար:

<select> թեգը սահմանում է ցուցակն ինքնին, իսկ <option> թեգը՝ դրա տարրերը:

```
<label for="city">Выберите город:</label>
<select id="city" name="city">
  <option value="moscow">Москва</option>
  <option value="spb">Санкт-Петербург</option>
  <option value="kazan">Казань</option>
</select>
```

disabled selected hidden

1. Աղյուսակի հիմունքներ

HTML-ում աղյուսակները ստեղծվում են <table> թեգի միջոցով: Աղյուսակի հիմնական կառուցվածքը բաղկացած է տողերից և բջիջներից.

Տողերը սահմանվում են <tr> թեգի միջոցով:

Տողերի բջիջները կարող են լինել վերնագիր կամ կանոնավոր.

Վերնագրի բջիջները սահմանվում են <th> պիտակի միջոցով:

Կանոնավոր բջիջները սահմանվում են <td> պիտակի միջոցով:

Հիմնական աղյուսակի օրինակ.

```
<table>
  <tr>
    <th>Имя</th>
    <th>Возраст</th>
    <th>Город</th>
  </tr>
  <tr>
    <td>Иван</td>
    <td>25</td>
    <td>Москва</td>
  </tr>
  <tr>
    <td>Мария</td>
    <td>30</td>
    <td>Санкт-Петербург</td>
  </tr>
</table>
```

Աղյուսակները կարող են ավելի բարդ լինել՝ օգտագործելով բջիջների միաձուլումը: Դրա համար օգտագործվում են colspan և rowspan ատրիբուտները.

colspan - տարածում է բջիջները հորիզոնական:

rowspan - ուղղահայաց տարածում է բջիջները:

```
<table border="1">
  <tr>
    <th rowspan="2">Имя</th>
    <th colspan="2">Возраст</th>
  </tr>
  <tr>
    <th>Молодой</th>
    <th>Старший</th>
  </tr>
  <tr>
    <td>Иван</td>
    <td>20</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Мария</td>
    <td>22</td>
    <td>32</td>
  </tr>
</table>
```

Styling աղյուսակները կարող են կատարվել CSS-ի միջոցով: Դուք կարող եք փոխել ֆոնի գույնը, տառատեսակը, բջիջի չափը և շատ ավելին: CSS-ի օրինակ աղյուսակը ձևավորելու համար.

```
<style>
  table {
    width: 100%;
    border-collapse: collapse;
  }

  th {
    background-color: #f2f2f2;
  }

  td, th {
    border: 1px solid #dddddd;
    padding: 8px;
  }
</style>
```

Դաս 4 - HTML & CSS

Հին դասին տնային առաջադրանքը կատարելիս ուշադրություն դարձրեցինք, որ մեկնաբանության դաշտը թույլ էր տալիս գրել միայն մեկ տողով, սակայն լինում են դեպքեր, երբ անհրաժեշտ է գրել մեկից ավելի տող: Այդ պարագայում անհրաժեշտ է կիրառել `textarea`: Հիմնական ատրիբուտներն են `rows`, `cols`, `placeholder`:

<section> -ը նախատեսված է բովանդակությունը բաժանելու համար ըստ թեմատիկ բաժինների: Ինչի համար են օգտագործում **<section>** -ը՝ այն օգտագործվում է կայքի էջի հստակ և պարզ կառուցվածք ստեղծելու համար: Բացի այդ կայքի տարբեր բաժինները անհրաժեշտ է առանձնացնել **<section>** թեգի միջոցով:

<nav> -ը նախատեսված է կայքում նավիգացիոն հղումներ ստեղծելու համար: Այն օգնում է որոնողական համակարգերին և այլ աջակցող տեխնոլոգիաներին ավելի լավ հասկանալ կայքի կառուցվածքը:

<article> -ը նախատեսված է ինքնաբավ բովանդակության ցուցադրման համար, օրինակ՝ նորության հոդվածներ, բլոգներ և այլն: Կարճ ասած ցանկացած ինֆորմացիա, որը կայքի էջի կոնտենթստից դուրս չի կորցնում իր իմաստային կշիռը:

<aside> -ը նախատեսված է կողմնակի կոնտենտի ցուցադրման համար, որը կապ ունի հիմնական կոնտենտի հետ, սակայն նրա հիմնական մաս չի հանդիսանում: Նպաստում է հասանելիությանը և SEO, ինչի շնորհիվ օգնում է որոնողական համակարգերին:

<div> - division դա HTML բլոկային տարր է, որը օգտագործվում է տարրերի խմբավորման և վեբ էջի կառուցվածքի ստեղծման համար:

**** — HTML ներտողային տարր է, որն օգտագործվում է բովանդակության փոքր մասեր ընդգծելու և ձևավորելու համար՝ առանց նոր տող անցնելու:

div, **span** համարվում են կոնտենտային տարրեր և կարող են պարունակել բազմաթիվ տողային և բլոկային տարրեր:

```
<div>
  <h1></h1>
  <p></p>
  <h3></h3>
  ...
</div>
<div>
  <span><p></p></span>
  <span><h1></h1></span>
  ...
</div>
<span>
  <div>
    <span>...</span>
  </div>
</span>
```

class="class1"-ը տարրին վերագրում է կլասի անուն: Կլասները օգտագործվում են ոճերի կասկադային աղյուսակում: Թույլատրվում է մեկ տեգին վերագրել մեկից ավելի կլասներ:

```
<div class="class1 class2 class3 ...">
    ...
</div>
```

ID -ն տարրին վերագրում է եզակի ատրիբուտ որը նույնպես կարող է օգտագործվել CSS ում

```
<div id="uniqueID">
    ...
</div>
```

CSS - մեկնաբանությունները սկսում են `/* comment */`

```
/*
Մեկնաբանություն / Comment
    Կոմենթում գրված տեքստը չի երևում և չի ազդում էջի տեսքի վրա
    CSS-ում կոմենթները օգտագործվում են միայն կոդի մեկնաբանելու
    և հասկանալու համար՝ կոդի ըմբերցելիությունը բարելավելու նպատակով:
*/
```

CSS -ը գրվում է նշելով այն թեգերը, որին վերաբերվելու է ոճը, իսկ ձևավոր փակագծերում գրվում է հատկությունները՝ իրենց դիմացը արժեքները:

```
div {
    width: 100px;
}

selector {
    property: value;
}
```

Ոճավորման կա երեք եղանակ՝ inline, head և առանձին ֆայլում:

Inline

```
<body>
    <div style="color: red;">
        ...
```

Head

```
<head>
    ...
    <style>
        * {
            margin: 0;
            padding: 0;
        }
    </style>
```

առանձին ֆայլում

```
<head>
...
<link rel="stylesheet" href="style.css">
```

CSS-ը կոչվում է "կասկադային" (Cascading) ոճավորման թերթիկ, քանի որ այն ունի կասկադային սկզբունք, ըստ որի՝ ոճերը կիրառվում են որոշակի հաջորդականությամբ և առաջնահերթությամբ: Կասկադի սկզբունքը կարգավորում է, թե որ ոճերն են ավելի գերակայում, երբ միևնույն տարրի վրա կիրառվում են տարբեր աղբյուրներից սահմանված ոճեր:

Կասկադային մեխանիզմը գործում է երեք հիմնական սկզբունքներով՝

1, Աղբյուրի սահմանման կարգ:

Սահմանվում են ըստ աղբյուրների տեսակի՝ (browser, css ֆայլ, head կոդ, inline կոդ

2. Ըստ selector-ի ստույգության՝

Որքան կոնկրետ է նշվում թե որ թեգին է վերաբերվում ոճը, այնքան այն առաջնահերթ է՝ օրինակ #div1 {} ավելի առաջնահերթ է քան div {}, որն էլ ավելի է քան * {}

3. Հայտարարությունների հերթականությամբ՝ Եթե միևնույն **selector**-ին հաջորդաբար վել է մեկից ավելի **հատկություն (property)** (օրինակ width: 150px;) ապա կիրառվում է ամենավերջին հայտարարվածը (հաշվի առնելով նաև վերոգրյալ դեպքերը):

Եթե ոճը պարունակում է **!important**, ապա այն գերակայում է մյուս բոլոր ոճերին՝ անկախ սելեկտորի ստույգությունից, աղբյուրի սահմանման կարգից կամ հայտարարությունների հերթականությունից:

Այսպիսով, եթե միևնույն հատկությունը (property) սահմանված է տարբեր արժեքներով և դրանցից մեկը նշված է **!important**, ապա կգործի այդ արժեքը, օրինակ՝

```
div {
    color: blue !important;
}

#divID {
    color: red;
}
```

CSS-ը ունի բազում հատկություններ, որոնցից մի մասի հետ արդեն ծանոթացել ենք՝

```
#divID {
    width: 100px;
    height: 100px;
    color: red;
    background-color: red;
    border: 1px solid red;
    border-collapse: collapse;
    ...
}
```

CSS-ի **display** հատկությունը կարգավորում է, թե ինչպես HTML տարրերը կդասավորվեն էջում: **display** հատկությունը ունի բազում արժեքներ, սակայն հիմնական երեքն են՝ **block**, **inline** և **inline-block**:

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

Ցուցադրվում է ինչպես բլոկային տարր, որը միշտ զբաղեցնում է ուղիղ հասանելի լայնությունը width և սկսվում են նոր տողից՝ **<p>**, **<h1>**, **<h2>**, **<div>** ...

Կարող ենք ոճավորել

```
div {  
    display: block;  
    width: ;  
    height: ;  
    margin: ;  
    padding: ;  
}
```

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

Ցուցադրվում է ինչպես տողային տարր, որը ցուցադրվում է մեկը մյուսի կողքին, եթե առկա է անհրաժեշտ լայնություն՝ ****, ****, **<u>**, **<a>** ...

```
div {  
    display: inline;  
    margin: ;  
    padding: ;  
}
```

* Լայնություն և բարձրություն չի ընդունում:

```
div {  
    display: inline-block;  
}
```

Տողային-բլոկային (inline-block) տարրեր են, որոնք կարող են ցուցադրվել նույն տողի վրա, ինչպես **inline** տարրերը, բայց պահպանում են իրենց բլոկային տարրերին բնորոշ հատկությունները, ինչպիսիք են լայնությունը և բարձրությունը:

```
div {  
    display: inline-block;  
    width: ;  
    height: ;  
    margin: ;  
    padding: ;  
}
```

CSS-ում չափերը նշելու համար օգտագործվում են տարբեր միավորներ, որոնք թույլ են տալիս ճկուն կերպով կարգավորել տարրերի չափերը՝ կախված Էկրանից, տառատեսակի չափից կամ այլ պարամետրերից՝

vmin – 1 միավոր **vmin**-ը համարժեք է Էկրանի լայնության կամ բարձրության փոքրագույն տոկոսի (1%):

vmax – 1 միավոր **vmax**-ը համարժեք է Էկրանի լայնության կամ բարձրության մեծագույն տոկոսի (1%):

Օրինակ եթե լայնությունը 1500px է, իսկ բարձրությունը 950px, ապա՝ $1vmin = 9.5px$, $1vmax = 15px$, իսկ $10vmin = 95px$, $10vmax = 150px$

Այս միավորներն օգտագործվում են, երբ ցանկանում ենք, որ տարրը փոփոխվի ըստ Էկրանի չափերի:

vw (viewport width) – համարժեք է 1% տեսադաշտի լայնությանը:

vh (viewport height) – համարժեք է 1% տեսադաշտի բարձրությանը:

Տեսադաշտը դա բրաուզերի այն հատվածն է, որը պատասխանատու է կայքերի ինտերակտիվ ելեմենտների ցուցադրման համար:

* Օգտագործվում է հիմնականում այն դեպքում, երբ տարրը պետք է որոշակի մաս կազմի Էկրանի չափերին:

```
div {
  width: 50vw;
  height: 20vh;
}
```

em – չափ է, որը կախված է տարրի ներկայիս տառատեսակի չափից: Օրինակ՝ եթե տարրի տառաչափը 16px է, ապա $1em = 16px$, եթե տվյալ տարրը չունի սահմանված տառաչափ, ապա հաշվի է առնվում նրա ծնող տարրի տառաչափը:

rem – չափ է, որը կախված է արմատային (root) տարրի տառաչափից (<html>): Օրինակ՝ եթե արմատային տարրի տառաչափը 16px է, ապա $1rem = 16px$:

Օգտագործվում են, երբ անհրաժեշտ է համեմատաբար ճկուն չափավորում՝ կախված տառաչափերից:

```
div {
  font-size: 2em;
}

p {
  font-size: 1.5rem;
}
```

px – հաստատուն չափ է, որը միանշանակ համապատասխանում է Էկրանին: Հաճախ կիրառվում է, երբ հարկավոր է ճշգրիտ վերահսկել տարրի չափերը:

Սակայն **px**-ը որոշ դեպքերում փոփոխական է տարբեր սարքերի միջև:


```
p {  
    font-size: 16px;  
}
```

ch – չափ է, որը հիմնված է «0» նիշի լայնության վրա՝ տառատեսակի համար: 1ch-ը 0 նիշի լայնությունն է, ինչը օգտակար է հիմնականում մոնոսպեյս (monospace) տառատեսակների համար: Մոնոսպեյս են այն տառատեսակները, որտեղ յուրաքանչյուր սիմվոլ ունի յույն լայնությունը՝ օրինակ `iii` և `www`:

Մոնոսպեյս տառատեսակներ են՝

Courier New, Consolas, Lucida Console, Roboto Mono, Source Code Pro

CSS-ի **font** հատկությունը կարգավորում է տեքստի տեսքը և ձևավորումը: Այն ներառում է մի քանի հատկություններ, որոնք սահմանում են տառատեսակը, չափը, գույնը, հաստությունը և այլն:

Font-family՝ օգտագործվում է տառատեսակը ընտրելու համար: Կարող է ներառել մի քանի տառատեսակ՝ պահուստային տարբերակներով:

```
p {  
    font-family: Arial, Helvetica, sans-serif;  
}
```

Font-size՝ կարգավորում է տեքստի չափը: Չափը կարող է նշվել միավորներով, ինչպես՝ px, em, rem, % և այլն:

```
p {  
    font-size: 24px;  
}
```

Font-weight՝ կարգավորում է տեքստի հաստությունը: Օգտագործվում են արժեքներ, ինչպիսիք են՝ normal, bold, կամ թվային արժեքներ (100-ից 900):

```
p {  
    font-weight: 900;  
}
```

Font-style՝ Կարգավորում է տեքստի ոճը, օրինակ՝ normal, italic, կամ oblique:

```
p {  
    font-style: italic;  
}
```

Font-variant` Օգտագործվում է փոքր գլխատառերի(մեծատառ) ձևի համար (small-caps): Այլ կերպ ասած փոքրատառերը կցուցադրվեն ինչպես մեծատառ, սակայն փոքր չափսերով:

```
p {  
    font-variant: small-caps;  
}
```

line-height` Կարգավորում է տեքստի տողի բարձրությունը` սահմանելով տողերի միջև տարածությունը:

```
p {  
    line-height: 1.5;  
}
```

CSS-ում կարելի է միանգամից գրել font հատկությունը` ներառելով վերը նշված հատկությունները`

```
p {  
    font: italic small-caps bold 16px/1.5 Arial, sans-serif;  
}
```

Դաս 5 - CSS selectors

CSS սելեկտորները օգտագործվում են HTML տարրերը ընտրելու և դրանց վրա ոճ կիրառելու համար: Սելեկտորները կարող են լինել **պարզ**, **բազմակի**, կամ **Էլ կոմբինացված**:

1. Պարզ սելեկտորներ

Տարրի սելեկտոր (element): Ընտրում է բոլոր տվյալ տարրերի օրինակները:

```
p {  
    color: blue;  
}
```

class-ային սելեկտոր (.class): Ընտրում է բոլոր այն տարրերը, որոնց վրա կիրառված է տվյալ կլասը:

```
.button {  
    background-color: green;  
}
```

id սելեկտոր (#id): Ընտրում է մեկ կոնկրետ տարր, որի ID-ն նշված է:

```
#submit {  
    background-color: green;  
}
```

2. Բազմակի սելեկտորներ

Սելեկտորները կարող են միանալ, որպեսզի ընտրեն միաժամանակ մի քանի տարրեր:

```
h1, h2, h3 {  
    color: red;  
}
```

3. Կոմբինացված սելեկտորներ

Descendant selector (ancestor descendant): Ընտրում է այն տարրերը, որոնք գտնվում են մյուս տարրի մեջ:

```
div p {  
    margin: 10px;  
}
```

Child selector (parent > child): Ընտրում է ուղղակի երեխային:

```
ul > li {  
    list-style-type: none;  
}
```

Adjacent sibling selector (previous + next): Ընտրում է այն եղբայրային տարրը, որը անմիջապես հաջորդում է:

```
h1 + p {  
    font-weight: bold;  
}
```

General sibling selector (previous ~ siblings): Ընտրում է բոլոր եղբայրական տարրերը, որոնք հաջորդում են:

```
H1 ~ p {  
    color: gray;  
}
```

Այժմ ուսումնասիրենք տեգերի որոշ հատկություններ՝

background հատկությունը նախատեսված է տեգի ֆոնի դիզայնը կարգավորելու համար՝

1, Պարզ ֆոնային գույն՝

```
div {  
    background-color: #eee;  
    /* Բաց մոխրագույն ֆոն */  
}
```

2, Ֆոնային պատկեր՝

```
div {  
    background-image: url("Image1.jpg");  
}
```

3, Ֆոնային նկարի դիրքի սահմանում՝

```
div {  
    background-image: url("Image1.jpg");  
    background-position: right top;  
}
```

4, Ֆիքսված ֆոն (պատկերն անշարժ է էջի շարժման ժամանակ)՝

```
div {  
    background-image: url("Image1.jpg");  
    background-attachment: fixed;  
}
```

5, Ֆոնի կրկնություն ըստ X, Y առանցքի

```
div {  
    background-image: url("Image1.jpg");  
    background-repeat: no-repeat; /* Կարելի է օգտագործել repeat-x կամ repeat-y */  
}
```

6, Հնարավոր է նաև իրականացնել գրառում համակցված օրինակով՝

```
div {  
    background: #ffffff url("Image1.jpg") no-repeat right top;  
}
```

7, Ֆոնային պատկերի մաշտաբավորում

```
div {  
    background: #ffffff url("Image1.jpg") no-repeat right top;  
    background-size: contain; /* cover */  
}
```

8, Ֆոնային շերտերի համադրություն

```
div {  
    background: url("Image1.jpg"), url("Image2.jpg");  
    background-blend-mode: multiply; /* hard-light, difference */  
}
```

CSS-ի margin հատկությունը սահմանում է տարածությունը էլեմենտի արտաքին կողմում՝ այլ էլեմենտների հետ ունեցած հեռավորությունը:

CSS-ի padding հատկությունը սահմանում է տարածությունը էլեմենտի ներքին կողմում՝ բովանդակության և եզրագծի (**border**) միջև ունեցած հեռավորությունը:

border հատկությունը սահմանում է էլեմենտի եզրագիծը՝ բովանդակության և տարածության շուրջը, ինչպես նաև սահմանում է դրա գույնը, լայնությունը և ոճը:

Float հատկությունը թույլ է տալիս տարրը հոսքավորել էջում՝ տեղադրելով այն ձախ կամ աջ կողմում, մինչդեռ պահպանում է բլոկային տարրերին բնորոշ որոշ առանձնահատկություններ:

```
<div style="float:left">  
    Some text left  
</div>  
<div style="float:right">  
    Some text right  
</div>  
<div style="float:none">  
    Some text none  
</div>
```

clear հատկությունը օգտագործվում է տարրերին **float** հատկությամբ ոճավորված տարրերի ազդեցությունից ազատելու համար: Երբ ինչ-որ տարր ունի float հատկություն, հաջորդող տարրերը կարող են «**հնուել**» նրա կողքով, ինչը երբեմն կարող է խաթարել էջի դասավորությունը: **clear**-ը թույլ է տալիս վերահսկել այդ վարքագիծը՝ ասելով, որ հաջորդող տարրը չպետք է «**հնուի**» float տարրի կողքով:

left: Արգելում է տարրին հոսքավորել float: left ունեցող տարրի կողքով:

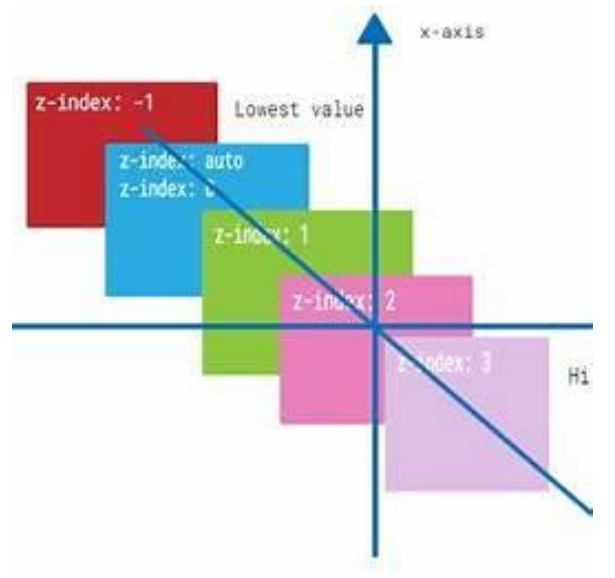
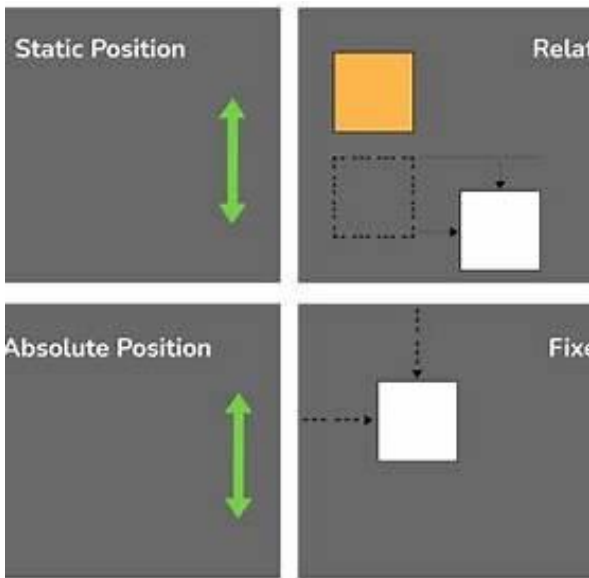
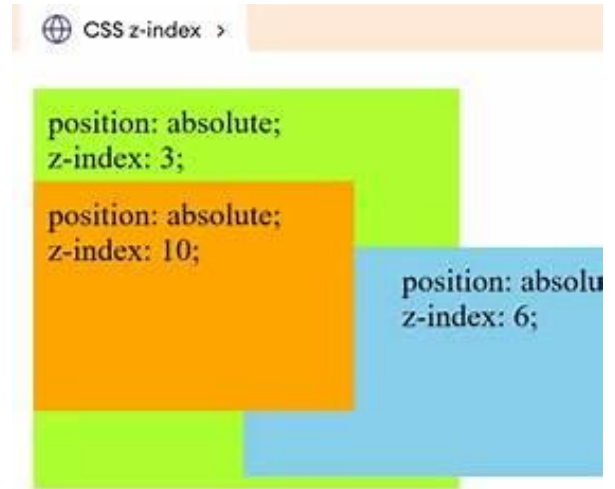
right: Արգելում է տարրին հոսքավորել float: right ունեցող տարրի կողքով:

both: Արգելում է տարրին հոսքավորել float: left և float: right ունեցող տարրերի կողքով:

<https://flukeout.github.io/>

Դաս 6 - Position, Z-index, Overflow

CSS-ում position, z-index, overflow հատկությունները կարևոր դեր են զբաղեցնում վեբ ծրագրավորման մեջ:



Position հատկությունը՝

Դիրքավորման (position) հատկությունը սահմանում է տարրի համար կիրառվող դիրքավորման մեթոդի տեսակը: Այն կարող է սահմանվել հետևյալ եղանակներով՝

Static: Տեղը դիրքավորվում է կայքի էջի բնականոն հոսքով: top, bottom, left, right, z-index հատկությունները ազդեցություն չունեն: Սա դիրքավորման լռելյայն (default) արժեքն է:

Relative: Տեղը դիրքավորվում է կայքի էջի բնականոն ընթացակարգով, սակայն ի տարբերություն ստատիկ դիրքավորման հնարավորություն է ընձեռնվում տեղափոխել Տեղը էջում top, left, right, bottom հատկություններով իր սկզբնական դիրքի համեմատ: Տեղափոխությունը չի ազդում հոսքի այլ տեղերի վրա: Այսպիսով relative-ը չի խախտում հոսքի դասավորությունը պահպանելով տեղի համար սահմանված տարածքը:

Absolute: Տեղը հեռացվում է կայքի էջի բնականոն հոսքից և էջի դասավորությունն մեջ տվյալ տեղի համար տարածք չի հատկացվում: Տեղը տեղակայվում է իրեն ամենամոտ տեղակայված ծնողի (եթե այդպիսին կա և եթե տեղը ունի մեկից ավելի ծնող <div1><div2><div3></div3></div2></div1> ապա div3 -ի համար որպես ծնող կվեկալի div2-ը): Absolute հատկությամբ տեղի վերջնական դիրքը որոշվում է top, bottom, left, right հատկություններով:

Absolute հատկությունը տեղադրում է տեղին այլ առանձին հոսքում, որի պաճառով z-index հատկության արժեքը չի վերագրվում ավտոմատ կերպով: Բացի այդ նաև հնարավորություն չի լինում ազդել հարևանների դիրքավորման վրա margin հատկության միջոցով:

Fixed: Տեղը հեռացվում է կայքի էջի բնականոն հոսքից: Տվյալ տեղի համար տարածք նույնպես չի հատկացվում: Տեղը տեղակայվում է հիմնական ծնող տեղի համեմատ (viewport): Fixed հատկությամբ տեղի վերջնական դիրքը որոշվում է top, bottom, left, right հատկություններով:

Sticky: Տեղը տեղադրվում է կայքի էջի բնականոն հոսքում: Տեղը դառնում է կաշուն երբ բավարարվում է top, bottom, left, right հատկություններով սահմանված պայմանները: Սակայն անհրաժեշտ է ուշադրություն դարձնել, որ կաշունությունը կաշխատի միմիայն անն դեպքում երբ կաշուն տեղի ծնող տեղը ունի սահմանված բարձրություն և չունի սահմանափակում overflow: auto:

z-index : Տեղին սահմանում է ցուցադրման կարգը` տվյալ տեղը ում պետք է ծածկի և ով պետք է ծածկի իրեն: Ավելի մեք z-index -ով տեղերը կարող են ծածկել ավելի փոքր ինդեքսով տեղերին: Բացառությամբ static հատկությունով տեղերի, այլ հատկությունով տեղերը կարող են ընդունել z-index հատկությունը:

Overflow: Տեղին սահմանում է ցանկալի վարքագիծ, որը պետք է կիրառվի բովանդակության չափերը սեփական չափի նկատմամբ գերազանցման դեպքում:

Overflow: visible: Տեղից դուրս գտնվող տարրերը տեսանելի են: Օրինակ եթե ունենանք մեծ տեքստ, սահմանափակ բարձրությամբ տեղի մեջ, ապա տեքստի այն հատվածը որը գտնվում է տեղից դուրս կցուցադրվի:

Overflow: hidden: Տեղից դուրս գտնվող պարունակությունը կթաքնվի և չի ցուցադրվի, սակայն կմնա հնարավորություն ծրագրային եղանակով պարունակությունը տեղաշարժելու:

Overflow: clip: Տեգից դուրս գտնվող պարունակությունը կտրվում է և չի ցուցադրվի ցանկացած եղանակով:

Overflow: scroll: Տեգին ավելացվում են հորիզոնական և ուղղահայաց սքրոլներ:

Overflow: auto: Տեգին կավելացվի հորիզոնական կամ ուղղահայաց սքրոլներ ըստ անհրաժեշտության:

Clip-path: Clip հատկության միջոցով կարելի է նշել, թե բացարձակ դիրքավորված տարրի որ մասը պետք է ցուցադրել: Մնացած մասերը թաքցվում են: օրինակ`

```
clip-path: polygon(0% 0%, 100% 0, 100% 100%, 50% 75%, 0% 100%);
```

Պետք է հաշվի առնել, որ չենք կարող տալ 100% ից մեծ չափս:

[Clippy — CSS clip-path maker \(bennettfeely.com\)](https://bennettfeely.com/clippy/)

visibility: visible: Տեգը տեսանելի է:

visibility: hidden: Տեգը տեսանելի չէ, սակայն իր տեղը կայքի էջի հոսքում պահպանվում է:

visibility: collapse: Աղյուսակի տարրը դադարում է ցուցադրվել չպահպանելով կայքի հոսքում իր տեղը: Այս հատկության արժեքը տրվում է միայն աղյուսակային տարրերին:

Դաս 7 - Pseudo-elements, Pseudo-classes

Clip-path: Clip հատկության միջոցով կարելի է նշել, թե բացարձակ դիրքավորված տարրի որ մասը պետք է ցուցադրել: Մնացած մասերը թաքցվում են: օրինակ՝

```
clip-path: polygon(0% 0%, 100% 0, 100% 100%, 50% 75%, 0% 100%);
```

Պետք է հաշվի առնել, որ չենք կարող տալ 100% ից մեծ չափս:

[Clippy — CSS clip-path maker \(bennettfeely.com\)](https://bennettfeely.com/clippy/)

Pseudo-elements (կեղծ տարրեր)-ները թույլ են տալիս ոճավորել տեգերի որոշ մասեր: Pseudo-element -ները Pseudo-class-ներից տարբերվում համար Pseudo-element -ները գրվում են կրկնակի վերջակետով: Նրանք աշխատում են միայն բլոկային տեգերի հետ: **/**/**

```
/* Ոճավորում է յուրաքանչյուր <p> տարրի առաջին տողը */
```

```
p::first-line {  
    color: blue;  
}
```

```
/* Ոճավորում է <p> տարրի առաջին տառը */
```

```
p::first-letter {  
    color: red;  
}
```

```
/* Ընտրված տեքստի ոճավորում */
```

```
::selection {  
    background-color: yellow;  
}
```

```
/* Հղումից առաջ ավելացնում է բովանդակություն */
```

```
a::before {  
    content: "↗ ";  
}
```

```
/* Հղումից հետո ավելացնում է բովանդակություն */
```

```
a::after {  
    content: " \>";  
}
```

Pseudo-class -ները կիրառվում են տեգերը կոնկրետ դեպքերում ոճավորելու համար: Pseudo-class -ները գրվում են վերջակետով:

```
/* Չայցելված հղումներ */
```

```
a:link {  
    color: blue;  
}
```

```

/* Այցելած հղումներ */
a:visited {
    color: rgb(98, 0, 128);
}

/* Մկնիկով սեզի վրան պահելիս */
a:hover {
    color: orange;
}

/* Տեզի վրա փիկ անելիս */
a:active {
    color: red;
}

```

Ինփուլթ չեքբոկսը նշելիս այլ գույնի սահմանում

```

input:checked {
    background-color: lightblue;
}

```

Այլ օրինակներ (:first-of-type, last-of-type)

```

/* Ցուցական առաջին դուստրը */
li:first-child {
    color: red;
}

/* Վերջին դուստրը */
li:last-child {
    color: blue;
}

/* Յուրաքանչյուր զույգ սեզը */
li:nth-child(even) {
    background-color: lightgray;
}

```

Scrollbar-ը ոճավորելու ժամանակ:

```

input:focus {
    background-color: lightyellow;
}

input:required {
    border: 2px solid red;
}

```

```

.main::-webkit-scrollbar {
    width: 12px;
}

.main::-webkit-scrollbar-thumb {
    background-color: darkgrey;
    border-radius: 10px;
}

```

Չեքրոքսի փոփոխության դեպքում

```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Menu (Checked) </title>
    <style>
        input[type=checkbox] {
            display: none;
        }

        .submenu {
            width: 200px;
            height: 200px;
            background-color: orange;
            display: none;
        }

        input[type=checkbox]:checked~.submenu {
            display: block;
        }
    </style>
</head>

<body>
    <label for="check">Click me</label>
    <input type="checkbox" id="check">
    <div class="submenu"></div>
</body>

</html>

```

Դաս 8 - Flexbox

Flex-ը նախատեսված է էլեմենտների արդյունավետ դասակարգման համար: **Flexbox**-ը (կարճ՝ **Flex**) հիմնականում օգտագործվում է էլեմենտների համար, որոնք դասավորված են մեկ ուղղությամբ՝ տողերով կամ սյուներով: Այս տեխնոլոգիայի վրա սկսել են աշխատել դեռ 2009 թվականից: 2012թ. ին ցուցադրվել է արդեն պատրաստի տարբերակը, իսկ 2015թ. ից արդեն հասանելի է եղել գրեթե բոլոր բրաուզերներում: **Flexbox**-ը ապահովում է ավելի մեծ ճկունություն, երբ մենք աշխատում ենք էլեմենտների դասավորության հետ: **Flexbox**-ը լուծում է շատ խնդիրներ, որոնք առաջանում են, երբ անհրաժեշտ է ճկունորեն կարգավորել տարրերի դիրքը և չափը:

Flexbox-ը հարմար է այն դեպքերի համար, երբ անհրաժեշտ է դասավորել էլեմենտները միայն մեկ ուղղությամբ՝ կամ տողերով (**horizontal row**), կամ սյուներով (**vertical column**). Այն թույլ է տալիս.

- Վերահսկել էլեմենտների չափսերը.
- Ավտոմատ լրացնել ազատ տարածքը.
- Դասավորել էլեմենտները հավասարապես կամ ճկուն կերպով:

Flexbox-ի հիմնական տարրը **flex-container**-ն է, որի մեջ պարունակվում են **flex-items**-ները:

- **Flex Container** – **flex** տեգ է, որը պարունակում է տեգեր որոնք պետք է դասավորվեն ըստ **flex** հատկությունների:
- **Flex Items** – կոնտեյնների մեջ գտնվող տարրերը, որոնք կդասավորվեն ըստ սահմանված կանոնների.

Flexbox-ի հիմնական հատկությունները՝

display: flex - դարձնում է տեգը **flex-container**, ինչը թույլ է տալիս ներսի տեգերին դասավորել ըստ flexbox կանոնների:

flex-direction - Նշում է թե որ ուղղությամբ պետք է դասավորվեն տեգերը:

```
.container {  
  flex-direction: row;  
}
```

row - Տարրերը դասավորվում են տողով՝ կողք կողքի:

row-reverse - Տարրերը դասավորվում են տողով՝ հակառակ հերթականությամբ:

column - Տարրերը դասավորվում են սյունով՝ իրար տակ:

column-reverse - Տարրերը դասավորվում են սյունով՝ հակառակ հերթականությամբ:

flex-wrap - Սահմանում է տեգերի վարքագիծը, երբ նրանց ընդհանուր երկարությունը կամ բարձրությունը գերազանցում է **flex-container**-ինը:

```
.container {  
  flex-wrap: wrap;  
}
```

nowrap- Տարրերը նոր տող (սյուն) չեն տեղափոխվում:

wrap- Տարրերը տեղափոխվում նոր տող (սյուն) եթե կա անհրաժեշտություն: (տողադարձ)

wrap-reverse- Տարրերը տեղափոխվում նոր տող (սյուն) հակառակ հերթականությամբ:

flex-flow - Քանի որ սովորաբար flex-container օգտագործելիս մշտապես կիրառվում են **flex-direction** և **flex-wrap**, դրա համար սահմանվել է **flex-flow** -ն որը միավորում է այս երկու հատկությունները:

```
.container {  
  flex-flow: row wrap;  
}
```

justify-content - Սահմանում է, թե ինչպես պետք է տարրերը տարածվեն գլխավոր առանցքի ուղղությամբ (հորիզոնական, եթե **row**, կամ ուղղահայաց, եթե **column**):

flex-start: Տարրերը դասավորվում են սկիզբից:

flex-end: Տարրերը դասավորվում են վերջից:

center: Տարրերը դասավորվում են կենտրոնում:

space-between: Տարրերի միջև հավասար տարածություն է սահմանվում, իսկ առաջին և վերջին տարրերը տեղակայվում են եզրերի մոտ:

space-around: Տարրերի միջև հավասար տարածություն է սահմանվում, իսկ եզրերի միջև՝ սահմանվում տարածության կեսը:

space-evenly: Տարրերի միջև հավասար տարածություն է սահմանվում, ներառյալ նաև եզրերը:

```
.container {  
  justify-content: center;  
}
```

align-items - Սահմանում է, թե ինչպես պետք է տարրերը տարածվեն գլխավոր առանցքի ուղղությամբ (հորիզոնական, եթե **column**, կամ ուղղահայաց, եթե **row**):

flex-start: Տարրերը դասավորվում են վերևում:

flex-end: Տարրերը դասավորվում են ներքևում:

center: Տարրերը դասավորվում են կենտրոնում:

stretch: Տարրերը զբաղեցնում են ողջ հասանելի բարձրությունը:

baseline: Տարրերը դասավորվում են ըստ բազային դիրքի:

align-content - Սահմանում է, տարրերի միջև ազատ տարածության բաշխումը, եթե այդ տողերը զբաղեցնում են քիչ բարձրություն քան:

flex-grow: հատկությունը սահմանում է, թե որքան պետք է տեղն աճի իր հարևանների համեմատ՝ եթե ազատ տարածք կա **flex-container**-ում: Այս հատկությունը ընդունում է դրական թվային արժեք, որը ցույց է տալիս աճելու գործակիցը:

```
.item {  
  flex-grow: 2;  
}
```

Այս դեպքում `.item` կլաս պարունակող տեգերը կաճեն 2 անգամ, քան այն տարրերը, որոնց սահմանել ենք `flex-grow 1`:

flex-shrink: հատկությունը թույլ է տալիս որոշել, թե որքան պետք է փոքրանա տարրը, երբ կոնտեյներում տեղը քիչ է: Այս հատկությունը նույնպես ընդունում է թվային արժեք: Եթե այն մեծ է, տարրը ավելի շատ կպակասի:

```
.item {
    flex-shrink: 1;
}
```

flex-basis: հատկությունը սահմանում է, թե որքա՞ն տարածք պետք է սկզբում զբաղեցնի տարրը մինչև նրա մեծացումը կամ փոքրացումը: Այն գործնականում աշխատում է որպես տարրի սկզբնական չափի սահմանիչ:

```
.item {
    flex-basis: 200px;
}
```

flex: կարճ գրառումն է, որը միավորում է երեք հատկությունները՝ `flex-grow`, `flex-shrink` և `flex-basis`-ը:

```
.item {
    flex: 1 0 100px;
}
```

Այս դեպքում տարրը կունենա **flex-grow**: 1, **flex-shrink**: 0 և **flex-basis**: 100px:

1 — տարրը աճում է ազատ տարածքի 1 մասով:

0 — տարրը չի փոքրանում, եթե տարածքը փոքրանում է:

100px — տարրի նախնական չափսը 100px է:

order հատկությունը թույլ է տալիս վերահսկել տարրերի հերթականությունը՝ անկախ նրանից, թե նրանք որ հերթականությամբ են գրված HTML-ում:

order-ի արժեքը ըստ լռելայն կարգի 0 է: Ավելի փոքր արժեքները նշանակում են ավելի վաղ, ավելի մեծ արժեքները՝ ավելի ուշ ցուցադրում:

```
.item {
    order: 2;
}
```

align-self-ը թույլ է տալիս վերահսկել մեկ կոնկրետ տարրի դիրքը իր կոնտեյների ներսում՝ անկախ մյուսների հետ հարաբերությունից:

Այս հատկությունը ընդունում է նույն արժեքները, ինչ `align-items`, բայց ազդում է միայն մեկ տարրի վրա:

```
.item {
    align-self: center;
}
```

Դաս 9 - Grid

CSS Grid Layout-ը դասավորության համակարգ է, որը հնարավորություն է տալիս երկու ուղղություններով (տողեր և սյուներ) ձևավորել ցանց: Ի տարբերություն Flexbox-ի, որը միակողմանի համակարգ է, Grid-ը երկկողմանի է և առավել ճկուն է մեծ նախագծերի համար:

```
.container {  
  display: grid;  
  grid-template-columns: 200px 1fr 200px;  
  grid-template-rows: 100px auto 100px;  
}
```

Օրինակ հին ժամանակներում կայքերը պատրաստելիս օգտագործում էին աղյուսակները՝ head, body, sidebar, footer -ը տեղաբաշխելու համար, հետո օգտագործում էին float: left, right, հետո **flex-ները**, մասնավորապես bootstrap4 -ը դա է օգտագործում, իսկ արդեն flex-ից հետո ստեղծվել է grid -ը որը բուն արդեն ցանցի համար է ստեղծված: Սկսած 2019թ -ից գրեթե բոլոր բրաուզերները արդեն ունակ են աշխատել grid -ի հետ: Սակայն դեռ մինչ այսօր կան որոշ բրաուզերներ, որոնք չեն կարողանում օգտագործել **Grid**: Բրաուզերների ցուցակը տեսնելու համար կարող ես օգտվել <https://caniuse.com/> կայքից, որոնման դաշտում գրելով grid:

Քանի որ հեռախոսի էկրանները փոքր են և մեծամասամբ հեռախոսի բրաուզերներում օբյեկտները դասավորվում են մեկը մյուսից հետո, այսինքն բլոկային տարբերակով, օգտագործվում է բլոկային տեգեր:

Տերմինաբանություն՝

Կոնտեյներ (container) - դա որոշ պարունակություն է, որը ունի display: grid հատկություն: Կոնտեյները ունի որոշ քանակությամբ տողեր, սյուներ և թույլ է տալիս կառավարել իր պարունակության վարքագիծը: Ի տարբերություն **Flex**-ի եթե մենք ցանկանանք տարբեր ռճավորումներ տանք տողերին, անհրաժեշտ չէ ստեղծել առանձին կոնտեյներներ տողերի համար:

Պարունակություն (Item) - Կոնտեյների դուստր էլեմենտներն են: Item-ները կարող են նույնպես ունենալ grid որոշ հատկություններ, որոնք կարող են ազդել իրենց դիրքավորման վրա:

```
<style>  
  div>div {  
    width: 150px;  
    height: 150px;  
    border: 1px solid black;  
    border-radius: 15px;  
    background-color: gray;  
  }  
  
  .container {  
    display: grid;
```

```

        grid-template-columns: repeat(4, 150px);
        grid-gap: 10px;
    }
</style>
<body>
    <div class="container">
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
    </div>
</body>

```

Grid-line - հիմնական կոնցեպտն է, որի շնորհիվ կառուցվելու է մեր հիմնական Grid-հատկությունները: Արտաքինապես կարծես մեր Grid-ը աղյուսակ է, սակայն իրչականում դա այդպես չէ, այն հատուկ ցանց է: Grid-line -ը պետք է կարողանանք պատկերացնել: Օրինակ վերևի օրինակում մենք ունենք 4 ուղղահայաց grid-line և 3 հորիզոնական:

Grid-Cell - **Grid-line** -ի արանքը ընգաժ տարածքն է: Այն պարտադիր չէ, որ պետք է լինի **Grid-item**, քանի որ կարող է լինել դեպքեր, որ այն մնա դատարկ, կամ էլ մեկ **Grid-cell** -ում տեղակայված լինի մեկից ավելի **Grid-item**-ներ:

Grid-Track - Տիրույթ է որոշ երկու զուգահեռ grid-line -րի արանքում, պարզ լեզվով ասած այն տող է կամ սյուն:

```

.yellow {
    background-color: yellow;
}
</style>
</head>
<body>
    <div class="container">
        <div class="item"></div>
        <div class="item"></div>
        <div class="item "></div>
        <div class="item yellow"></div>
        <div class="item yellow"></div>
        <div class="item yellow"></div>
    </div>
</body>

```

Grid-range - Գրիդ տիրույթ: Տիրույթները կարելի է անվարկել, ու հետագայում իրմել տիրույթներին իրենց անուններով և այլն: Դրա մասին իհարկե կծանոթանանք:

```

<body>

```



```

<div class="container">
  <div class="item yellow"></div>
  <div class="item yellow"></div>
  <div class="item "></div>
  <div class="item yellow"></div>
  <div class="item yellow"></div>
  <div class="item "></div>
</div>
</body>

```

Ամփոփենք՝ ընդհանուր GRID տերմինները որ մենք թվարկեցինք 6 էին՝ **Կոնտեյներ (container)**, **Պարունակիչ (Item)**, **Grid-line**, **Grid-Cell**, **Grid-Track** և **Grid-range**:

grid- ի հասարակ օրինակ օգտագործելով միայն մեկ կլասս

```

.container {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 1rem;
}

```

Fr - fraction, доля, բաժին

```

.container > div {
  min-width: 150px;
  min-height: 150px;
  border: 1px solid black;
  background-color: gray;
}

```

```

.container {
  display: grid;
  gap: 1rem;
}

```

Այժմ փորձենք փոխազդել grid-ի հետ item-ների միջոցով՝

```

.container > div {
  min-width: 150px;
  min-height: 150px;
  border: 1px solid black;
  background-color: gray;
}

.container {
  display: grid;
  gap: 1rem;
}

```

```

.item1 {
    grid-column: 1 / 2;
}

.item2 {
    grid-column: 2 / 3;
}

```

Մեր փոփոխությունները ավտոմատ փոխադրել են նաև այն item-ների վրա, որոնք մենք չենք նշել: Դիմելով grid-column 1/2, 2/3 մենք կարծես թե նշել ենք որ մեր ցանցը պետք է ունենա ուղղահայաց 3 grid-line: Կարող ենք շարունակել, տալով item3 -ին **grid-column: 3 / 4**; Մենք դեռ տալիս ենք հատկություններ միայն grid-item -ներին: Եկեք ստեղծենք ևս մեկ grid-container:

```

<div class="container">
    <div class="item5"></div>
    <div class="item6"></div>
    <div class="item7"></div>
</div>

```

Այժմ մենք աշխատում ենք ինչպես աղյուսակների հետ: Աղյուսակները ինչպես հիշում ենք կարող ենք այնպես անել, որ երկու կամ ավելի բջիջ միավորվեն ինչպես մեկը: Նմանատիպ ֆունկցիոնալ կա նաև grid -ում:

```

.item5 {
    grid-column: 1 / 2;
    grid-row: 1 / 3;
}

.item6 {
    grid-column: 2/3;
    grid-row: 1/2;
}

```

Ինչպես տեսնում ենք մենք գրել ենք ընդամենը 2 style բայց արդեն մեր աղյուսակը ստացել է այլ տեսք: Մեր item5 -ը փաստորեն զբաղեցնում է 1-ից 2-րդ սյուները և 1-ից 3-րդ տողերը: Նման հայտարարման մեթոդով հեշտ է նաև փոփոխելու item-ների հերթականությունը, ենթադրենք եթե մենք ցանկանանք, որ item5 -ը նախորդի item5-ին:

Մենք օգտագործում ենք կոմբինացված տարբերակը, սակայն կարող էինք օգտագործել՝ **grid-row-start: 1; grid-row-end: 3**; սակայն առաջին տարբերակով հայտարարելը ավելիօ հարմար և կարճ է:

Կարող ենք նաև հստակ չսահմանել ավարտը, օրինակ՝

```

.item:first-child {
    grid-column: 2 / span 2;
}

```

Սկսվում է 2-րդ ից և զբաղեցնում 2 սյուն

Ենթադրենք ցանկանում ենք արհմիտիվ կայքի կարկաս հավաքենք՝

```

<body>
  <div class="container">
    <header>
      <h1>Header</h1>
    </header>

    <article>
      <h2>Title</h2>
      <p>Lorem500</p>
    </article>
    <aside>
      <h3>Mi ban</h3>
      <blockquote> <!-- цитата-->
        Inch vor meki gracy
      </blockquote>
    </aside>
  </div>
</body>

```

```

<style>
  header {
    background-color: gold;
  }
  article {
    background-color: aqua;
  }
  aside {
    background-color: bisque;
  }
</style>

```

Օրինակ հեռախոսային տարբերակների համար գրեթե արդեն պատրաստ է, քանի որ փոքր էկրանների դեպքում հիշում ենք որ անհրաժեշտ է կիրառել բլոկային տարբերակով: Ուղղակի կարող ենք ավելացնել`

```

.container {
  display: grid;
  gap: 10px;
}

```

```

header {
  background-color: gold;
  grid-column: 1/3;
  grid-row: 1/2;
}

```

Կարող ենք նաև տեղափոխել

```
article {  
    background-color: aqua;  
    grid-column: 2/3;  
    grid-row: 2/3;  
}
```

```
div.item$*5  
grid-template-columns:  
grid-template-rows:
```

Սյուների համար շաբլոն սարքելիս՝

```
.item {  
    min-width: 100px;  
    min-height: 100px;  
    background-color: blue;  
}  
  
.container {  
    display: grid;  
    gap: 10px;  
    grid-template-columns: 200px 500px 150px;  
}
```

Տողերի համար սարքելիս՝ եթե տողերի քանակը չատ է մեր թվարկածից, ապա այն կընդունի լռելայն արժեքը

```
.container {  
    display: grid;  
    gap: 10px;  
    grid-template-columns: 200px 500px;  
    grid-template-rows: 200px 500px;  
}
```

Եթե նույն չափը պետք է կրկնել մի քանի անգամ կարող ենք օգտագործել repeat՝

```
grid-template-columns: repeat(3, 150px);
```

Եթե ցանկանում եմ հայտարարել միայն առաջին և վերջին item-ի չափը՝

```
grid-template-columns: 150px 1fr 150px;
```

Ֆրակցիան զբաղեցնում է ազատ տիրույթը: Այն համագործաբցում է այլ ֆրակցիաների հետ, օրինակ՝

```
grid-template-columns: 2fr 1fr 150px;
```

Ստանում ենք հարաբերակցություն, իր ֆունկցիոնալով նման է %-ային արժեքներին:

Սովորաբար ֆրակցիաները տողերի համար չեն տալիս այլ օգտագործում են **auto**:

Նաև gap -ին կարող ենք տալ երկու արժեք, օրինակ ինչպես margin, padding -ին՝

```
gap: 10px 25px;
```

Flex-ում գիտենք որ կա **order**, նույն հատկությունը կա նաև **gap** -ում:

Դատարկ տիրույթներ

```
<div class="container">
  <header>header</header>
  <article>
    <h2>Vernagir</h2>
    <p>lorem 150</p>
  </article>
  <aside>
    mi ban
  </aside>
  <footer>
    footer
  </footer>
</div>
```

```
<style>
  header {
    background-color: gray;
  }
  article {
    background-color: pink;
  }
  aside {
    background-color: red;
  }
  footer {
    background-color: aquamarine;
  }
</style>
```

Տանք Grid հատկություն՝

```
.container {
  display: grid;
  gap: 10px;
}
```

Այժմ ուսումնասիրենք

```
grid-template-areas: none;
```

Եթե այն չենք օգտագործում, ապա իր արժեքը հավասար է none

```
header {
  background-color: gray;
  grid-area: header;
}
article {
```

```

        background-color: pink;
        grid-area: article;
    }
    aside {
        background-color: red;
        grid-area: aside;
    }
    footer {
        background-color: aquamarine;
        grid-area: footer;
    }

```

Կարծես թե ամեն ինչ փչացավ, բայց դա նրանից է, որ`

```

.container {
    display: grid;
    gap: 10px;
    grid-template-columns: repeat(3, 1fr);
    grid-template-areas:
        "header"
        "article"
        "aside"
        "footer";
}

```

Կարելի է սահմանել նաև այսպես`

```

grid-template-areas:
    "header header header"
    "article article aside"
    "footer footer footer";

```

Այժմ սարքենք դատարկ տիրույթ` օգտագործելով կետը

```

.container {
    display: grid;
    gap: 10px;
    grid-template-columns: repeat(4, 1fr);
    grid-template-areas:
        "header header header header"
        "article article . aside"
        "footer footer footer footer";
}

```

Կա նաև ավելի հեշտացված գրելու մեթոդ`

```

grid-template: none | grid-template-rows | grid-template-columns

```

```

grid-template: "header header header header" 50px
               "article article . aside"

```

```
"footer footer footer footer" 50px / 1fr 1fr 50px 1fr;
```

Սակայն այն օգտագործվում է եթե չունենք area -ներ

Այժմ ուսումնասիրենք հավասարեցումները`

```
header {
    background-color: gray;
}

article {
    background-color: pink;
}

aside {
    background-color: red;
}

footer {
    background-color: aquamarine;
}

.container {
    display: grid;
    gap: 10px;
    grid-template: auto / 1fr 1fr 1fr; (auto կամ անեմ 500px 300px)
}
```

Ու կարող ենք օգտագործել justify-items ինչպես դա արել ենք flex-ում:

```
justify-items: start;
justify-items: center;
justify-items: end;
justify-items: stretch; // լռելյայն տարբերակ
```

Նույնը կա նաև align-items -ի համար:

Կա նաև կոմբինացված տարբերակը

```
place-items: center center;
```

Այժմ մենք չունենք սահմանաձևվում ներ կոնտեյների լայնության կամ երկարության համար, ենթադրենք մեզ դա անհրաժեշտ է`

```
.container {
    display: grid;
    width: 700px;
    gap: 10px;
```

```
grid-template: auto / 100px 250px 120px;  
border: 1px solid black;  
place-items: center center;  
}
```

Այժմ ունենք հավելյալ տարածք որը չի օգտագործվում:

```
.container {  
  display: grid;  
  width: 700px;  
  gap: 10px;  
  grid-template: auto / 100px 250px 120px;  
  border: 1px solid black;  
  place-items: center center;  
}
```

Ու ցանկանում ենք դիրքավորել մեր ցանցը օգտագործելով ազատ տիրույթը, նման պարագայում մեզ օգնության է գալիս արդեն justify-content

```
justify-content: space-between;
```

Նույն եղանակով նաև ուղղահայաց դիրքավորման համար՝

```
align-content: start;
```

Կա նաև կոմբինացված տարբերակը՝ սկզբից align-content, justify-content

```
place-content: center space-evenly;
```

Ֆլեքսից հիշում ենք նաև որ կարող ենք հայտարարել անհատական հատկություններ կոնկրետ item-ներին: Այստեղ նույնպես կարող ենք տալ justify-self, align-self

```
aside {  
  background-color: red;  
  align-self: start;  
}
```

Սակայն Grid-ով աշխատելիս խորհուրդ չի տրվում օգտագործել մնացած եղանակով ֆիքսված չափսեր, քանի որ բուն Grid-ի իմաստը կորում է:

Ադապտիվ Grid

Ունենք՝

```
<div class="container">  
  <div class="item item1">1</div>  
  <div class="item item2">2</div>  
  <div class="item item3">3</div>  
  <div class="item item4">4</div>  
  <div class="item item5">5</div>  
  <div class="item item6">6</div>  
  <div class="item item7">7</div>  
</div>
```



```
<style>
  html, body {
    margin: 0;
  }

  .container {
    background-color: gainsboro;
    padding: 1.5rem;
    display: grid;
    gap: 10px 15px;
  }

  .item {
    background-color: orange;
    border-radius: 15px;
    padding: 1rem;
  }
</style>
```

Սահմանում ենք միևնույն և մաքսիմալ չափերը, կարող ենք սահմանել պիքսելներով, ֆրակցիաներով

```
.container {
  background-color: gainsboro;
  padding: 1.5rem;
  display: grid;
  gap: 10px 15px;

  grid-template-columns: 200px 1fr minmax(100px, 200px);
}
```

Օրինակ եթե շատ մեծ տողեր ունենանք, ապա այն կարող է սահմանից դուրս գալ, դա բացառելու համար օգտագործում ենք`

```
grid-auto-rows: minmax(100px, auto);
```

Այն սահմանում է բոլոր նոր տողերի համար միևնույն 100 պիկսել

Օրինակ ենթադրենք մեզ պետք է որպեսզի մեր ֆուտերը եթե կոնտենտը քիչ է կպած լինի ներքևի հատվածին`

```
<div class="item item1">1</div>
<div class="item item2">2</div>
<div class="item item3">3</div>
```

```
.container {
  background-color: gainsboro;
  padding: 1.5rem;
  display: grid;
```

```
gap: 10px 15px;
grid-template-rows: 100px minmax(100px, 1fr) 50px;
```

Սա հիմա չի աշխատում քանի որ մենք դիվ ենք հայտարարել ու նրա բարձրությունը գիտենք որ հավասար է իր պարունակության բարձրությանը եթե սահմանված չի, այդ իսկ դեպքում պետք է տանք նաև `height: 100vh;` որպեսզի բարձրությունն սահմանենք: Այսինքն եթե `grid container` -ը չունի սահմանված բարձրություն այն եղանակը աշխատելու է որպես `auto`:

```
height: 100vh;
/* padding: 1.5rem; */
```

Հիշում ենք նաև որ `flex`-ում մենք կարող էինք փոփոխել ուղղությունը, այսինքն սահմանել `row`, `column`: Մնամ \$ունկցիոնալ կա նաև այստեղ`

```
display: grid;
gap: 10px 15px;
grid-template-columns: 200px 1fr minmax(100px, 200px);
grid-template-rows: minmax(100px, auto) 100px 200px;
grid-auto-flow: row;
```

Վերադառնանք մեր սյունակների ադապտիվ ցուցադրման օրինակին`

```
grid-template-columns: repeat(3, 1fr)
```

Ինքը կրկնում էր մեր փոխանցված մեծությունը նշված քանակի անգամ: Բացի փոխանցվող կրկնությունների քանակը մենք կարող ենք փոխանցել նաև մեթոդը:

```
grid-template-columns: repeat(auto-fit, minmax(200px, 250px))
```

```
grid-template-columns: repeat(auto-fill, minmax(200px, 250px))
```

`Auto-fit` -ի տրամանաբությունը`

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr))
```

Ինքը փորձում է զբաղեցնել ողջ ազատ տիրույթը` մեծացնելով մեր `item`-ների չափերը:

```
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr))
```

`Auto-fill` -ի տրամանաբությունը`

Ինքը տեսնում է թե քանի հատ կարելի է լրացնել մաքսիմալ տվյալ տողում`

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr))
```

Մենք չենք ուսումնասիրենք `grid` հատկությունը, այն կատարում է նույն գործողությունները ինչ `grid-template`:

Մենք կարող ենք տալ անհատական վարքագիծ ցանկացած `grid-item` ներին: Տալ վարքագիծ ընդհանուր `grid-container` -ին:

[Responsive Periodic Table with CSS Grids index.html - nodebox - CodeSandbox](#)

Դաս 10 - Photoshop, Figma, GIT

Photoshop ներածություն`

Adobe Photoshop-ը հզոր գրաֆիկական խմբագիր է, որն օգտագործվում է պատկերների խմբագրման, գրաֆիկայի ստեղծման և լուսանկարների ոճավորման համար: Այն լայնորեն կիրառվում է դիզայնի, լուսանկարչության, վեբ մշակման և այլ ոլորտներում:

Նոր CC 2023-ը բերում է արտադրողականության բարելավումներ, ուժեղացված AI գործիքներ և նոր հնարավորություններ:

Figma ներածություն`

Figma-ն ամպային գրաֆիկական ինտերֆեյս է, որը նախատեսված է նախագծման, ոճավորման համար: Այն շատ հարմարավետ գործիք է մեծ պրոեկտների վրա թիմային աշխատանքի համար:

Git ներածություն`

Git-ը ծրագրերի վերսիաների կառավարման համակարգ է, որը թույլ է տալիս հետևել և վերահսկել ֆայլերի փոփոխությունները, համակարգել աշխատանքները նախագծերի վրա տարբեր թիմերի հետ:

Դաս 11 - CSS Animation

CSS3-ի անիմացիան հնարավորություն է տալիս կայքի տարրերին ավելացնել դինամիկ անիմացիաներ: Անիմացիան տարմեր կադրերի համախումբ է, որոնք կրկնվում են որոշակի ժամանակահատվածում: Անիմացիան կարելի է կիրառել գրեթե բոլոր HTML տեգերի վրա բացառությամբ `<head>`, `<meta>`, `<title>`, `<Script>`, `<link>`, `<style>`:

animation-name - այն անիմացիայի անվանումը, որը պետք է կիրառվի տվյալ տեգին:

@keyframes - անիմացիայի ֆունկցիան, որը որոշում է թե ինչ տեսակի փոփոխություններ պետք է իրականացվեն անիմացիայի ժամանակ:

animation-duration - այն ժամանակահատվածը, որի ընթացքում պետք է կատարվի անիմացիան:

animation-timing-function - Անիմացիայի արագության փոփոխությունները:

animation-delay - Անիմացիան սկսելու ուշացման ժամանակը:

animation-iteration-count - Անիմացիայի կրկնության քանակը:

animation-direction - Անիմացիայի կրկնության ուղղությունը:

animation-play-state - Անիմացիայի ցուցադրումը և դադարը:

animation-fill-mode - @keyframes-ի որոշակի օբյեկտների ոճավորման հաջորդականությունը:

CSS Transition-ը թույլ է տալիս էլեմենտին շարժվել որոշակի ժամանակ, որոշակի քանակությամբ, և այդ ընթացքում փոփոխել իր հատկությունները: Սա տեղի է ունենում pseudo-class-ով հայտարարված որևէ իրադարձության ընթացքում (օրինակ՝ **:hover**):

Այն ունի հիմնական հատկություններ՝

transition-property - Գործողություն կատարելու ժամանակ փոփոխվող հատկությունների անվանումները:

transition-duration - Գործողության տևողությունը:

transition-timing-function - Անիմացիայի արագության փոփոխությունները:

transition-delay - Դադարի ժամանակը՝ արժեքը եթե կա:

Sprite-ը համարվում է շարժական պատկերներ ստանալու մեթոդներից մեկը: Սա մի գրաֆիկական պատկեր է, որն իր մեջ պարունակում է անիմացիայի բոլոր կադրերը, և տարբեր հատկություններով կարելի է կառավարել այդ կադրերի աշխատանքը՝ արագությունը, շարժը, կադրերի քանակը:

@font-face կանոնակարգը թույլ է տալիս ընտրել ոչ ստանդարտ տառատեսակներ՝ բեռնել դրանք, անվանում որոշել և միացնել նախագծին:

Դաս 12 - Bootstrap, font awesome

Library (գրադարան) - ենթածրագրերի և օբյեկտների հավաքածու է, որը հեշտացնում և արագացնում է ծրագրավորման նախագծերի իրականացման գործընթացը: Որոշ դեպքերում դրանք անվանվում են նաև մոդուլներ: Ծրագրի գրադարան ասելով հասկանում ենք՝ նախապես մշակված ծրագրեր այն հատվածների համար, որոնք հաճախ են հանդիպում ծրագրի ստեղծման փուլում:

Այն զբաղեցնում է լոկալ տիրույթում 200 - 300KB եթե տեղադրենք որպես \$այլ:

Պլագին - bootstrap 5 & Font Awesome Snippets

Bootstrap-ի տերմինների և կլասների մասին գրականությունը գտնվում է Documentation բաժնում: Այստեղ ենթաբաժիններով տարանջատված են տարբեր էլեմենտներին վերաբերվող կլասները:

Container class-ը համարվում է Bootstrap-ի ամենագլխավոր հատկություններից մեկը և շատ անհրաժեշտ է բլոկային էլեմենտների օգտագործման համար: Այս կլասը էլեմենտին տալիս է ֆիկսված լայնության չափ: Որպիսի էլեմենտը ձգվի մինչև Էկրանի աջ - ձախ եզրեր անհրաժեշտ է օգտագործել container-fluid կլասը:

Bootstrap-ի կոնտեյներում կիրառվում է ցանցերի մեթոդը՝ տողեր և սյուներ: Ցանցը բաժանված է 12 սյուների, տողերի քանակը կախված է կայքից տարողությունից՝ փոփոխվող է: Յուրաքանչյուր տողի պարունակությունը գրվում է row կլասով էլեմենտի մեջ:






```
<div class="container-fluid">
  <div class="row">
    <!-- Կոնտենտ, սյուներ -->
  </div>
</div>
```

Container - ունի ֆիքսված լայնություն, որը փոփոխվում է կախված Էկրանի չափսերից, թողնելով կողմերում բաց միջակայքեր

Container-fluid - միշտ զբաղեցնում է 100% Էկրանի լայնությունը:

Բուլստրապում ցանցը բաղկացած է 12 սյունակներից: Սյունակների կիրառումը իրականավորվում է ըստ թվանշանների, այսինքն՝ col-12 այս տեղը զբաղեցնելու է 12 սյունակից 12-ը իսկ col-4 -ը զբաղեցնելու է 12 ից 4-ը

Bootstrap-ում կիրառվում է em և rem չափման միավորները, իսկ px-ը container-ի լայնության չափերի համար: Տարբեր սարքավորումների էկրանների չափերը հետևյալն են՝

xs	Extra small <576px	 portrait mobile
sm	Small ≥576px	 landscape mobile
md	Medium ≥768px	 portrait tablets <i>navbar collapse</i>
lg	Large ≥992px	 landscape tablets
xl	Extra large ≥1200px	 laptops, desktops, TVs

Բուժստորապում հատկությունները կարելի է կոմբինացնել, այսինքն՝

Col-md-4 col-xs-4 և այլն

Եթե մի տողում սյուների քանակը 12-ից մեծ լինի ապա վերջին էլեմենտը կիջնի ներքև: Եթե ցանկանում ենք տողի սյուներ բաժանել և էլեմենտների մի մասն տեղափոխել ներքև, կիրառում ենք w-100 կլասը:

```
<div class="container-fluid">
  <div class="row">
    <div class="col-6 col-md-4">.col-6 .col-md-4</div>
    <div class="col-6 col-md-4">.col-6 .col-md-4</div>
    <div class="col-6 col-md-6">.col-6 .col-md-6</div>
  </div>
</div>
```

Եթե մի տողում սյուների քանակը 12-ից մեծ լինի ապա վերջին էլեմենտը կիջնի ներքև: Եթե ցանկանում ենք տողի սյուներ բաժանել և էլեմենտների մի մասն տեղափոխել ներքև, կիրառում ենք w-100 կլասը:

```
<div class="container">
  <div class="row">
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="w-100"></div>
    <div class="col">col</div>
    <div class="col">col</div>
  </div>
</div>
```

Bootstrap-ում մենք կարող եք վերահսկել սյունակների դիրքավորումը՝ կիրառելով համապատասխան դասեր: Ստորև ներկայացված են հիմնական դասերը և դրանց կիրառման եղանակները, որոնք կօգնեն ձեզ դիրքավորել բովանդակությունը row-ի կամ սյունակի ներսում:

align-items-* դասեր

Այս դասերը կառավարում են բովանդակության ուղղահայաց դիրքավորումը row-ի ներսում: Դրանք կարգավորում են սյուների դասավորությունը ուղղահայաց առանցքով:

- **align-items-start** — Սյուները կտեղադրվեն row-ի վերին մասում:
- **align-items-center** — Սյուները կտեղադրվեն ուղղահայաց կենտրոնում:
- **align-items-end** — Սյուները կտեղադրվեն row-ի ներքևում:

```
<div class="row align-items-center" style="background-color: red; height: 200px;">
  <div class="col">Սյուն 1</div>
  <div class="col">Սյուն 2</div>
  <div class="col">Սյուն 3</div>
</div>
```

align-self-* դասեր

Այս դասերը կիրառվում են առանձին սյուներին row-ի ներսում՝ թույլ տալով նրանց ուղղահայաց դիրքավորումը սահմանել անկախ մյուս սյուներից:

- **align-self-start** — Սյունը կտեղադրվի վերևում:
- **align-self-center** — Սյունը կտեղադրվի ուղղահայաց կենտրոնում:
- **align-self-end** — Սյունը կտեղադրվի ներքևում:

```
<div class="row" style="height: 200px; background-color: lightgray;">
```

```

<div class="col align-self-start">Սյուն 1 (վերևում)</div>
<div class="col align-self-center">Սյուն 2 (կենտրոնում)</div>
<div class="col align-self-end">Սյուն 3 (ներքևում)</div>
</div>

```

justify-content-* դասեր

Այս դասերը կառավարում են սյուների հորիզոնական դասավորությունը **row**-ի ներսում, տարածելով կամ կենտրոնացնելով սյուները:

- **justify-content-start** — Սյուները կտեղադրվեն ձախ կողմում:
- **justify-content-center** — Սյուները կտեղադրվեն կենտրոնում:
- **justify-content-end** — Սյուները կտեղադրվեն աջ կողմում:
- **justify-content-around** — Հավասար տարածություն կլինի սյուների շուրջ:
- **justify-content-between** — Սյուները կտեղադրվեն երկու կողմերում, իսկ միջին տարածությունը կբաշխվի հավասարապես:

```

<div class="row justify-content-around">
  <div class="col-4">Սյուն 1</div>
  <div class="col-4">Սյուն 2</div>
  <div class="col-4">Սյուն 3</div>
</div>

```

text-center text-left text-right text-stretch text-nowrap	տեքստի տեղափոխումը հորիզոնական ուղղությամբ
text-success text-info text-warning ...	տեքստի տառերի գույնավորում bootstrap-ի ստանդարտ գույներով
bg-success bg-info bg-warning ...	տեքստի տիրույթի գույնավորում bootstrap-ի ստանդարտ գույներով

Աղյուսակներ

```
<table class="table table-striped table-bordered table-hover table-dark">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Սնունդ</th>
      <th scope="col">Ազգություն</th>
      <th scope="col">Տարի</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Հայկ</td>
      <td>Հայ</td>
      <td>25</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Աննա</td>
      <td>Հայ</td>
      <td>30</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Արամ</td>
      <td>Հայ</td>
      <td>28</td>
    </tr>
    <tr>
      <th scope="row">4</th>
      <td>Սեդա</td>
      <td>Հայ</td>
      <td>22</td>
    </tr>
  </tbody>
</table>
```

```
<div class="container mt-5">
  <h2>Bootstrap Image Demonstration</h2>

  <h4>img-fluid</h4>
```

```


<p>Այս նկարը փոփոխվում է, երբ կայքի չափերը փոփոխվում են:</p>

<h4>img-thumbnail</h4>
    
    <p>Այս նկարը ունի եզրեր, որոնք զօսվում են:</p>

<h4>rounded-circle</h4>
    
    <p>Այս նկարը ունի շրջանաձև ծև:</p>
</div>
```

Դաս 13 - Media

Մեդիա հարցումները դա CSS ֆիլտրներ են, որոնք թույլ են տալիս իրականացնել որոշակի ոճավորում, որը կիրականացվի էջի կոնկրետ էկրանի չափսերի դեպքում:

Մեդիա հարցումները հաշվի են առնում ինչպես բրաուզերի viewport-ի այնպես էլ սարքավորման բարձրությունը և լայնությունը: Բացի բարձրությունից հեռախոսների համար կարող ենք նաև հաշվի առնել նրա էկրանի դիրքավորումը՝ **portrait** (ուղղահայաց) ու **landscape** (հորիզոնական):

Մեդիա հարցումը գրվում է @media բանալի բառի օգնությամբ: Մեդիաում կարող ենք օգտագործել չափման միավորներ՝ px, vh, vw: Media հարցումները պետք է գրվեն չիշտ հերթականությամբ՝ մեծ էկրանները վերևում, փոքր էկրանները ներքևում:

Min-width, max-width

Եթե ցանկանում եք կիրառել ոճեր միայն այն դեպքում, երբ լայնությունը փոքր է կամ հավասար է որևէ արժեքի, օգտագործեք **max-width**:

Եթե ցանկանում եք կիրառել ոճեր միայն այն դեպքում, երբ լայնությունը մեծ է կամ հավասար որևէ արժեքի, օգտագործեք **min-width**:

Media օպերատորներ

Օգտագործելով **and**, կարող եք միավորել մի քանի պայմաններ մեկ մեդիա հարցման մեջ: Օրինակ, եթե ցանկանում եք կիրառել ոճերը միայն այն դեպքում, երբ էկրանի լայնությունը մեծ է 768px-ից և էկրանը գտնվում է **landscape** (հորիզոնական) դիրքում:

```
/* Այս ոճերը կկիրառվեն, երբ էկրանի լայնությունը 768px-ից մեծ է և դիրքը հորիզոնական է */
.container {
    background-color: lightgreen;
    font-size: 20px;
}
```

not օպերատորը շրջում է հարցման արդյունքը: Օրինակ, եթե ցանկանում եք կիրառել ոճերը բոլոր էկրանների վրա, որոնք **չեն բավարարում** որևէ հատուկ պայման:

```
@media not all and (max-width: 600px) {

    /* Այս ոճերը կկիրառվեն, եթե էկրանի լայնությունը 600px-ից մեծ է */
    .container {
        background-color: lightblue;
    }
}
```

```
}
```

only օպերատորը օգտագործվում է բրաուզերներում համապատասխանությունն ապահովելու համար: Դա նշանակում է, որ հարցումը կկիրառվի միայն այն դեպքում, երբ այն ճանաչվում է բրաուզերի կողմից: Այն հիմնականում օգտագործվում է հին բրաուզերներից ապահովության համար, բայց ժամանակակից բրաուզերներում այն հազվադեպ է անհրաժեշտ:

```
@media only screen and (min-width: 1024px) {  
  
    /* Այս ռեգլը կկիրառվեն միայն էկրանների վրա, երբ լայնությունը 1024px կամ  
ավելի է */  
  
    .container {  
        background-color: lightcoral;  
        font-size: 22px;  
    }  
}
```

Այս մեդիա հարցումը կկիրառվի միայն այն դեպքում, եթե սարքը էկրան է և լայնությունը 1024px կամ ավելի է:

, (ստորակետ) օպերատորը CSS-ում մեդիա հարցումների դեպքում գործում է որպես "կամ"՝ թույլ տալով միավորել մի քանի մեդիա հարցումներ: Այսինքն՝, եթե օգտագործում եք ստորակետ, ապա ռեգլը կկիրառվեն, եթե գոնե մեկ հարցումը ճշմարիտ է:

```
@media screen, print {  
  
    /* Այս ռեգլը կկիրառվեն և՛ էկրանին, և՛ տպման ժամանակ */  
    body {  
        font-size: 16px;  
        color: #333;  
    }  
}
```

```
@media (max-width: 600px), (min-width: 1200px) {  
  
    /* Այս ռեգլը կկիրառվեն, եթե էկրանի լայնությունը 600px-ից փոքր է կամ 1200px-ից մեծ */  
    .container {  
        background-color: lightblue;  
    }  
}
```

Կարող եք միավորել տարբեր օպերատորներ՝ ստեղծելով բարդ մեդիա հարցումներ:

```
@media only screen and (min-width: 768px) and (max-width: 1200px) and (orientation: portrait) {  
  
    /* Այս ռեկերը կկիրառվեն միայն այն ժամանակ, երբ էկրանը portrait դիրքում է, լայնությունը՝ 768px-ից մեծ և 1200px-ից փոքր */  
    .container {  
        background-color: lightyellow;  
        font-size: 18px;  
    }  
}
```

Այս մեդիա հարցումը կկիրառվի միայն այն ժամանակ, երբ էկրանն ունի **portrait** դիրք, լայնությունը 768px-ից մեծ է, բայց 1200px-ից փոքր:

Էկրանից բացի կան նաև այլ տիպի media տիպեր՝

all- տարածվում է բոլոր սարքերի վրա (կանխադրված):

print - օգտագործվում է տպագրության ժամանակ:

screen - օգտագործվում է համակարգչի և շարժական սարքերի էկրանների համար:

speech - խոսքի սինթեզատորների համար:

Կան նաև ավելի քիչ տարածված մեդիա տեսակներ.

projector- պրոյեկտորների համար:

tv - հեռուստացույցների վրա ցուցադրելու համար:

