FrontINT mavzulari

Created @September 11, 2022 6:36 PM

HTML

Head va body qismining farqi nima ?

HTTP so’rovlarda bo’lgani kabi, Body sahifani asosiy kontentlarini o’z ichiga oladi, head esa sahifa haqidagi ma’lumotlarni, masalan meta teglar, stillar, title va hkz. Texnik tarafdan qarasak, HTML tepadan pastga qarab o’qib keladi, head esa birinchi bo’lib o’qiladi.

!doctype nima ?

!doctype sahifani browser tomonidan qanday formatda o’qilishi kerakligini belgilab beradi. Hozirda biz ishlatadigan format HTML, ammo boshqa formatlar ham mavjud. Masalan XHTML, Framesets.

Teg turlari: Inline va Block teg farqi nima ?

Harbir HTML elementning default display xossasi mavjud. Bu teg turlariga qarab, inline yoki block bo’lishi mumkin. Block elementni Inline element ichida yozish mumkin emas va unday qilish xato.

Block elementlarga misol: div, h1, p, ul.

Inline elementlarga misol: span, a, b, i, img.

Meta teglar orqali nimalar qila olamiz ?

Meta teglar sahifa haqida ma’lumotlarni ulash uchun ishlatiladi. Shuningdek ularni ishlatishdan yana bir asosiy maqsad SEOni rivojlantirish bo’ladi. Ya’ni, masalan Googledan qidirganda sahifangizni qidiruvda topish uchun Google aynan shu teglardagi ma’lumotlardan foydalanadi(*yana boshqa ma’lumotlardan ham*).

CSS fayllarni ulash tartibi qanday ?

Dastur talablaridan kelib chiqib harxil yo’llari bor, masalan:

1. CSS fayllar head tegi ichida.

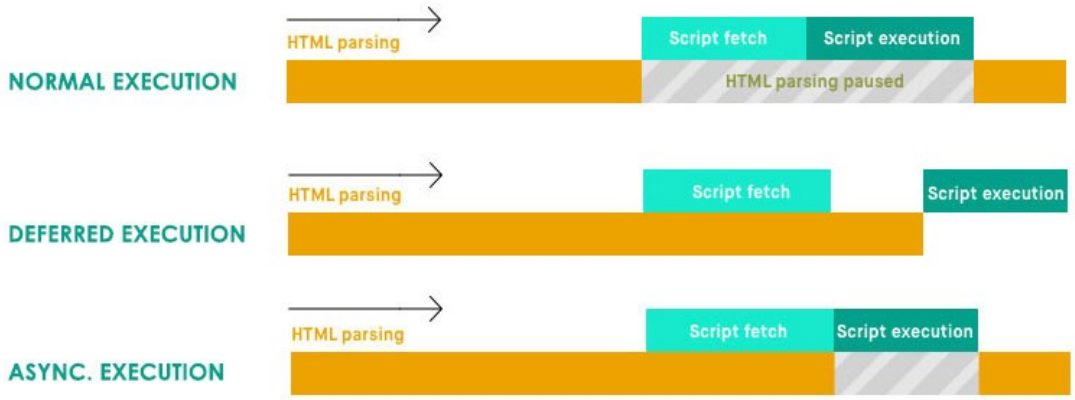
2. CSS fayllar kerakli qismini head tegi ichida(Critical CSSni inline yozish). Qolganini asinxron yuklash.

```
<style> /*  Critical CSS */ </style>
<link rel="stylesheet" href="style.css" media="print" onload="this.media='all'">
```

▼ JS fayllarni ulash tartibi qanday ?

1. JS fayllar body tegi oxirida yoki head tegi ichida.
2. JS fayllar kerakli qismini body tegi oxirida. Qolganini asinxron yuklash. Yoki hammasini asinxron yuklash. Masalan:

```
// Async yuklanishi bilan ishlaydi.HTML o'qib bo'linishini(parsing) kutib turmaydi.
// Ishlashni boshlaganda parser bloklanadi.
<script async src="script.js">
// Defer yuklangandan so'ng HTML to'liq o'qib bo'linishini kutib turadi.
// Ishlashni boshlaganda parser bloklanmaydi.
<script defer src="script.js">
```



▼ HTMLda Accessibility tushunchasi nima ?

Dasturni boshqa texnologiyalar to’g’ri o’qishini oson qilish bu Accessibility. Masalan screen readers, search engine va hkz. Asosiy maqsadlardan biri, nogiron insonlarga qulayliklar yaratish. Prinsiplari(by WAI):

1. Perceivable (Hamma saytdagi ma’lumotlarga ega bo’la olishi. Ko’r inson eshita olishi, kar inson ko’ra olishi, texnik muommolari bor inson boshqa

yo’llar orqali o’qiy olishi…)

2. Operable (Sayt boshqaruvi hamma uchun mos bo’lishi.)
3. Understandable (Sayt barcha uchun tushunarli bo’lishi.)
4. Robust (Sayt hozirgi va keyin kelishi mumkin bo’lgan qurilmalarda deyarli birhil ishlashi)

To’liq guideline: https://www.w3.org/TR/WCAG20/

Qanday qilib saytni Accessibility qismini oshirish mumkin ? Quyidagilar orqali:

1. Alt teglar (***<img src=”url” alt=”Rasm haqida ma’lumot”>***)
2. Semantik elementlar (*keyingi savolda*)
3. aria atributlar (*keyingi savolda*)
4. video/audio captions, autocomplete inputs, using keyboards va hkz.

To’liq list linki: romeo.elsevier.com/accessibility_checklist/

Screen reader extension: ChromeVox

▼ Semantik elementlar nima ?

Semantik elementlar - nomidan odam va kompyuter tushunadigan qilib yozilgan elementlardir. Masalan, <header><footer><main><section><nav> va hkz. Ularning asl ustunliklari quyidagilar:

1. O’qish va tushunishga oson.
2. Accessibilityni oshiradi.

▼ HTMLda aria atributlari o’rni qanday ?

Aria(Accessible Rich Internet Applications) attributlar saytni ko’zi ojiz insonlar ishlatishi oson bo’lishi uchun ishlatiladigan attributlar to’plami. Screen readerlar aria atributlarni dasturni ko’zi ojiz insonlarga to’g’ri va tushunarli o’qib berish uchun foydalanishadi.

Short english tutorial: https://www.youtube.com/watch?v=0hqhAIjE_8I
Aria atributlari to’plami: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Attributes

▼ Data-* atributlari nima ?

Ushbu atribut HTML elementga o’zimiz xohlagan attribut nomi ostida qiymat berishga yordam beradi. So’ng u qiymatni JavaScript va CSSda olib ustida ishlashimiz mumkin.

```
<!DOCTYPE html>
    <html>
      <head>
        <style>
          li::before {
            content: attr(data-author);
            background: red;
            margin: 10px;
          }
        </style>
      </head>

      <body>
        <ul>
          <li data-author="Author 1" id="book_1">Book 1</li>
        </ul>

        <script>
          const article = document.querySelector("#book_1");
          alert(article.dataset.author);
        </script>
      </body>
    </html>
```

▼ Canvas va SVG farqi nima ?

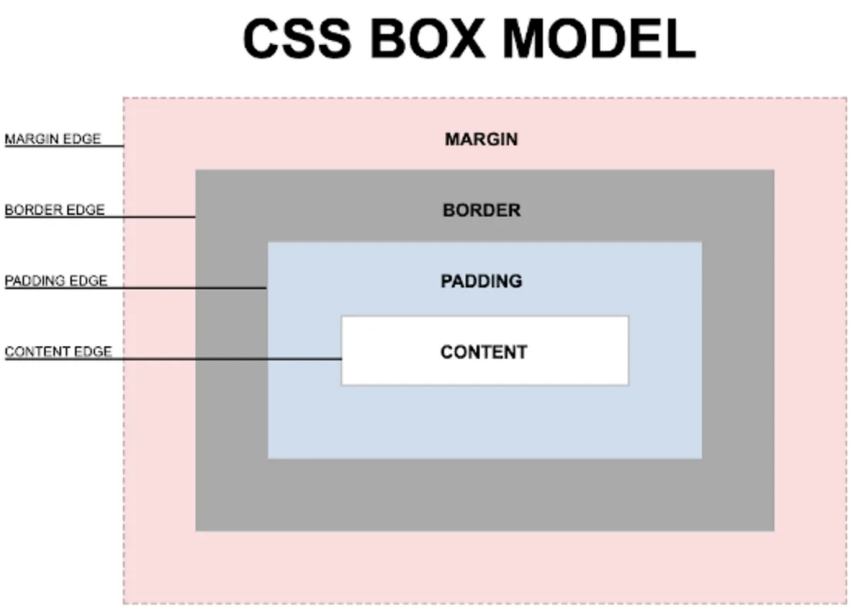
SVG - vector-based. Yuqori sifatli, ammo og’irroq.

Canvas - pixel-based. Sifati past, ammo yengil.

▼ CSS

▼ CSS Box model nima va nimalarni o’zini ichiga oladi ?

Har bir HTML element CSS Box modelga ega, uning ichiga quyidagilar kiradi(*Devtools orqali ham ko’rsa bo’ladi*):

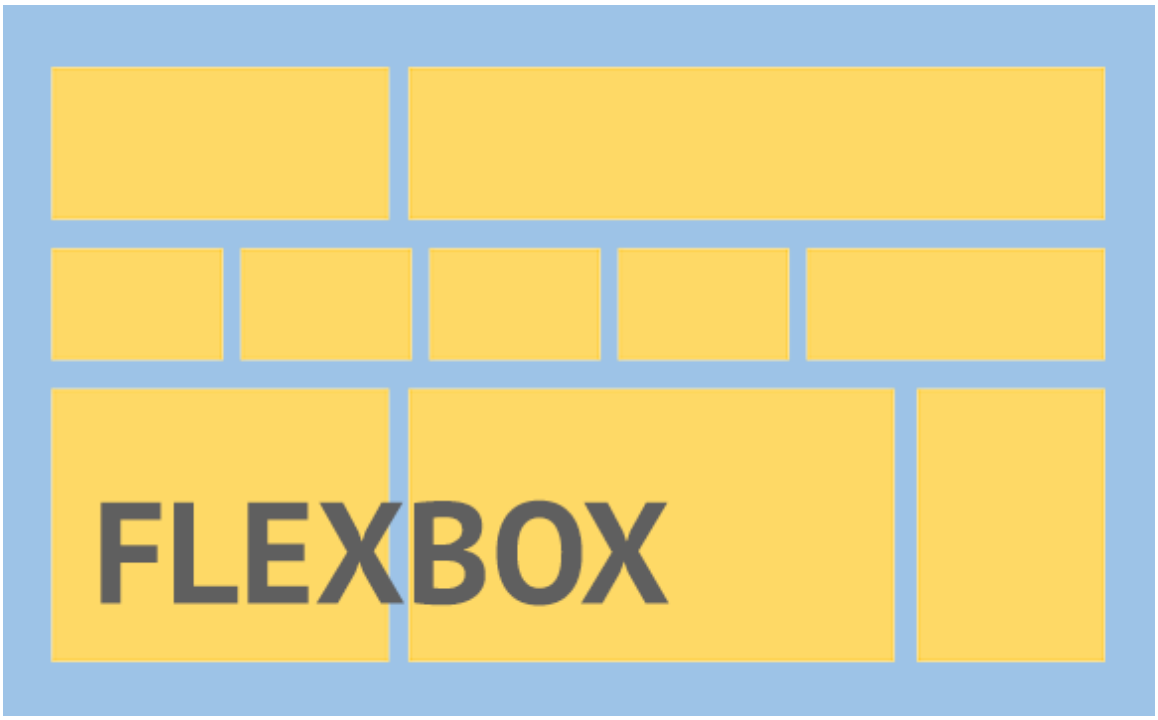


▼ CSSda display xossasining block, inline va inline-block qiymatlari farqlari nimada ?

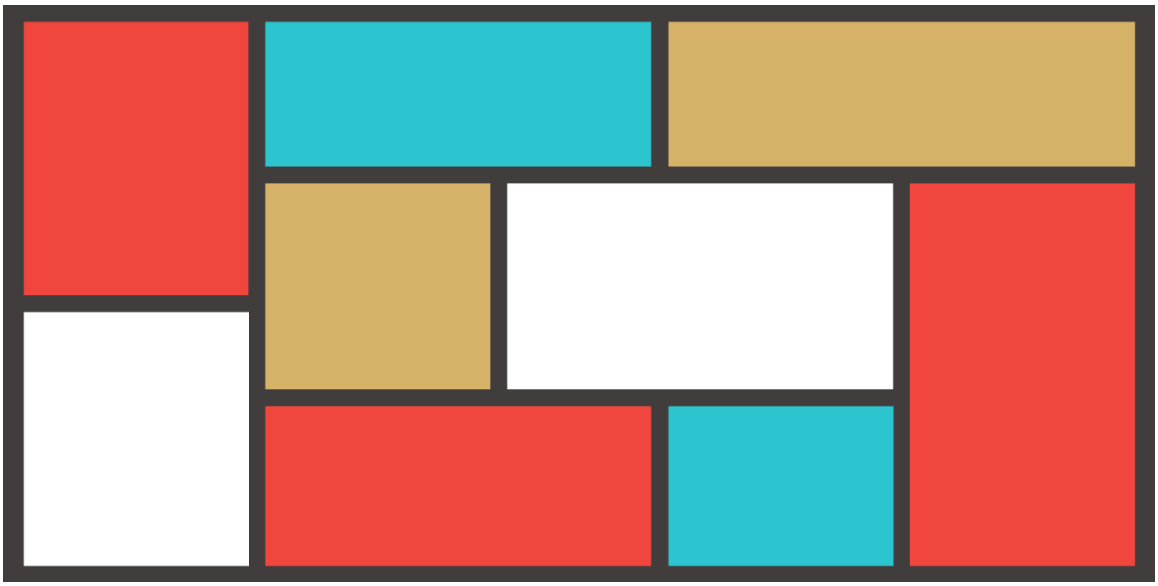
1. block - imkoni yetgancha uzunlikni oladi, yangi qator talab qiladi.
2. inline - imkon qadar eng kichik uzunlikda bo'ladi, width/height berib bo'lmaydi.
3. inline-block - imkon qadar eng kichik uzunlikda bo'ladi, width/height berib bo'ladi.

▼ Elementlarni joylashtirishda float, flexbox va grid ustunlik va kamchiliklari nimada ?

Flexbox - one-dimensional layoutlar uchun(row yoki column) . Ko'proq list va listsimon layout yasashda ishlatiladi. Masalan:



Grid - two-dimensional layout'lar uchun(row va column bir vaqtda). Murakkab shaklli layoutlar uchun. Masalan:



O'rganish uchun resurs:

Flexbox froggy o'yini: flexboxfroggy.com

CSS Grid Garden o’yini: cssgridgarden.com

▼ Qaysi position xossalarini bilasiz, absolute va relativening farqlari nimada ?

Eng mashxurlari, static(*default*), absolute, fixed, relative, sticky.

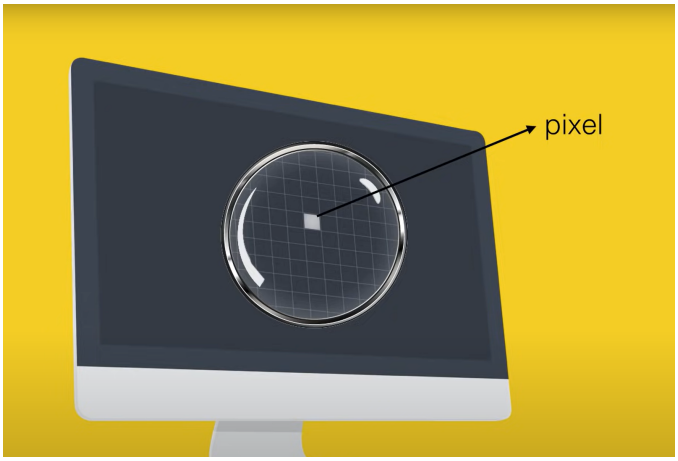
Absolute - o’zining birinchi static bo’lmagan ajdod elementiga qarab joy oladi. Unday element bo’lmasa o’z turgan joyiga nisbatan oladi.

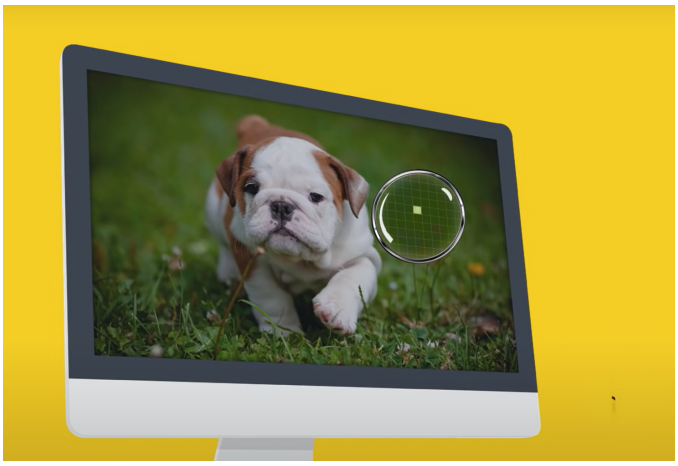
Relative - O’z turgan joyiga nisbatan joy oladi.

▼ Qaysi CSS unitlarni bilasiz va farqlari nimada ?

CSS da eng ko’p ishlatiladigan unitlar: px, em, rem, vw, vh

1. px - picture element (pix = picture, el = element). Istalgan ekranlarni tashkil etuvchi mayda zarrachalar, pixellar. (Absolute unit)





2. em - Elementning o’zini yoki ota tegini font-size o’lchamiga nisbatan olinadi. Masalan ota element font o’lchami 20px bo’lsa, unga em ishlatsangiz, 1em = 20px, 1.5em = 20px * 1.5 = 30px (Relative unit)
3. rem - Root elementning font-size o’lchamiga nisbatan oladi. (<html> tegi root element). Masalan, <html> tegiga 30px font-size bersangiz, ichidagi barcha teglarga 2rem = 60px bo’ladi. Default qiymat 16px. (Relative unit)
4. vw - viewport width. Browserning ekrani kengligiga nisbatan oladi. 1vw desak, ekranning kengligining 1%iga teng.
5. vh - viewport height. Browserning ekrani uzunligiga nisbatan oladi. 1vh desak, ekranning uzunligining 1%iga teng.

▼ Pseudo elementlar qaysilar ?

Pseudo elementlar - ma’lum CSS selector oldiga qo’shimcha ravishda qo’shish mumkin bo’lgan va shu tanlangan elementning ma’lum qismlarni tanlab olish uchun ishlatiladigan kali so’zlar. Masalan mashxurlari, ::after, ::before, ::first-line, ::first-letter, ::selection, ::placeholder, :nth-child(n), :last-child, :not(selector)

▼ Media query va uning xossalari haqida gapiring.

Media queries - dasturimizni harxil qurilmalar uchun moslashuvchanligini oshirish uchun ishlatiladi. U o’z ichiga qurilma turi(Media type) va qurilma xususiyati(width, height)ni olib, aynan shu shartlarga mos kelgan qurilmalarga kod yozish imkonini beradi.

Media type’lar quyidagilar:

1. all - barcha qurilmalar uchun.
2. print - printerlar uchun.
3. screen - Kompyuter, smartfonlar va tabletlar uchun.
4. speech - screenreaderlar uchun.

Media queries ga misol:

```
@media screen and (max-width: 992px) {
div {
background-color: red;
}
}
```

▼ CSSda Specificity Hierarchyni tushuntiring.

Specificity Hierarchy - CSSda HTML elementni tanlab olib, stil berishdagi ketma-ketlik.

- **Inline styles** - Example: <h1 style="color: pink;">
- **IDs** - Example: #navbar
- **Classes, pseudo-classes, attribute selectors** - Example: .test, :hover, [href]
- **Elements and pseudo-elements** - Example: h1, :before

Manba: https://www.w3schools.com/css/css_specificity.asp

▼ Bootstrap qanday ishlaydi ?

Bootstrap - qayta ishlatish uchun tayyor bo'lgan HTML, CSS va JS kodlardan tashkil topgan kutubxona. Asosiy qismi CSSda tayyor klasslar yaratishdan iborat. Bu dasturda CSS kod juda kam ishlatgan holda kod yozishga yordam beradi. Ammo, kichikroq dasturlarga tavsiya qilinmaydi. Chunki bunda, Bootstrapda bor ko'p narsadan foydalanilmaydi va ortiqcha joy oladi.

▼ CSSda dasturni optimizatsiya qilish usullari bormi ?

CSSda dasturni ishlashini yaxshilash va tez ishlashini ta'minlash uchun quyidagi maslahatlar beriladi:

1. Stillarni qismlarga ajratish va Critical bo'lmaganlarini asinxron yuklash.

2. Animatsiyalarni iloji boricha Transform, opacity kabi tez ishlaydigan xossalarda qilish.
3. DOMga ta’sir qiladigan animatsiyalardan qochish.
4. Selectorlarni oddiyroq qilish.

Ko’proq maslahatlar uchun: sitepoint.com/optimizing-css-performance

▼ CSS preprocessor nima va unga qaysilar kiradi ?

CSS preprocessor - bu bizga o’zining sintaksida kod yozishga imkon yaratib, natijani browser tushunadigan toza CSS kodga aylantirib beradi. Maqsad CSSda yo’q bo’lgan yangi va foydali xususiyatlarni bizga taklif qilish, masalan mixin, function, nesting selector va hkz. Ularga misol qilib, SCSS, LESS, Stylus va boshqalar kiradi.

▼ SCSS'ning ustunlik va kamchiliklari haqida gapiring.

SCSS bizga toza kod yozish va qayta takrorlashdan qochish(DRY) . Katta dasturlarda kodni boshqarishni va maintainance’ni osonlashtiradi. Bu natijalarga Mixin, Variables, Selector inheritance, functions va shu kabi xususiyatlari orqali erishiladi.

Kamchiliklari esa, SCSS’da generatsiya qilgan fayllar oddiy CSS’da yozilganidan kattaroq bo’lishi mumkin. Debug qilish ham oddiy CSS’ga qaraganda qiyinroq bo’ladi. Ammo, katta dasturlarda SCSS’ning kamchiligidan foydasi ko’proq bo’ladi.

▼ CSS va SCSS’da o’zgaruvchilar farqi ?

SCSS variable - runtime’da oddiy CSS qiymat sifatida chiqadi va uni o’zgartirib bo’lmaydi.

CSS variable - runtime-da ham o’zgartirsa bo’ladi. JavaScript orqali ham o’zgartirsa bo’ladi.

▼ Bir necha xil temali dasturni qanday yozish usulini bilasiz ?

Usullar ko’p. Eng ko’p ishlatiladiganlaridan biri CSS variable orqali qilish usuli. Bunda asosiy ranglar variablega olinadi va boshqa rangga o’zgarsa JS orqali hamma joyda o’zgaradi.

Misol: codesandbox.io/s/m45wj6xz68?from-embed=&file=/index.html:1732-1820

▼ SCSS best practices haqida gapiring.

1. Kodni partial fayllar ishlatish orqali qismlarga ajratish. (_variables.scss, _mixins.scss)
2. @use ishlatish. (@import o’rniga) Chunki, @use ishlatilgan holatda, bu faylni qayerda nechi marta ishlatsangiz ham, u birmarta import bo’ladi.
3. Ko’p ishlatilgan xossalarni Variable, mixin kabi qayta ishlatish mumkin bo’lgan joylarda saqlash.
4. Mixin’lardan to’g’ri foydalanish. Chunki ular copy/paste kabi shunchaki kodning takrorlanishi xolos.

```
/* Good example */
@mixin rounded-corner($arc) {
    -moz-border-radius: $arc;
    -webkit-border-radius: $arc;
    border-radius: $arc;
}
/* Bad example */
@mixin cta-button {
    padding: 10px;
    color: #fff;
    background-color: red;
    font-size: 14px;
    width: 150px;
    margin: 5px 0;
    text-align: center;
    display: block;
}
```

5. Placeholder’lardan foydalanish. Mixin’dan farqli o’laroq, DRY’ga amal qiladi.

```
%bg-image {
    width: 100%;
    background-position: center center;
    background-size: cover;
    background-repeat: no-repeat;
}

.image-one {
    @extend %bg-image;
    background-image:url(/img/image-one.jpg");
}

.image-two {
```

```
    @extend %bg-image;
    background-image:url(/img/image-two.jpg");
}

/* COMPILED */
.image-one, .image-two {
    width: 100%;
    background-position: center center;
    background-size: cover;
    background-repeat: no-repeat;
}

.image-one {
    background-image:url(/img/image-one.jpg") ;
}

.image-two {
    background-image:url(/img/image-two.jpg") ;
}
```

6. Hisoblashlar uchun Function’lardan foydalanish

```
@function calculate-width ($col-span) {
    @return 100% / $col-span
}

.span-two {
    width: calculate-width(2); // spans 2 columns, width = 50%}

.span-three {
    width: calculate-width(3); // spans 3 columns, width = 33.3%}
```

7. Nesting’da me’yorni bilish. (3 ta darajada Nesting’dan oshmaslik)

https://csswizardry.com/2016/02/mixins-better-for-performance/

▼ JS

▼ O'zgaruvchilar turlari va farqlari haqida gapiring.

JavaScriptda var, let va const o’zgaruvchilari mavjud. ES6 dan avval varning o’zi edi.

const - bu o’zgarmas. Block-scoped. Hoisting ishlaydi, ammo xatolik beradi.

let - o’zgaruvchi. Block-scoped. Hoisting ishlaydi, ammo xatolik beradi.

var - o’zgaruvchi. Function/Local-scoped. Hoisting ishlaydi (qiymati undefined) Global e’lon qilinsa, Window obyektiga yoziladi. (let va const unday emas). Birhil nom bilan qayta e’lon qilish ishlaydi(redeclaring). (let va const xatolik beradi)

▼ Hoisting nima ?

Hoisting - JSda mavjud bo’lgan default xususiyat bo’lib, u barcha declaration’larni alohida o’qib, ularni shu scope ichida eng tepaga ko’taradi, bu orqali declaration’larni teparoqda ham ishlatsa bo’ladi. Asosan funksiyalar bilan ishlaganda foyda beradi.

Declaration’larga misol - let, const, var, function, class

▼ TDZ (Tempororal dead zone) nima ?

Bu let va const’ning shu scope boshidan to ularni e’lon qilingan qatorigacha bo’lgan vaqt oraliqi.

Masalan,

```
if(true) {
   // TDZ start
   console.log(x); // x is dead
   alert(1);
   let x = 1; // TDZ end
 }
```

▼ Ma'lumotlar tiplari haqida gapiring.

JavaScript’da ma’lumotlar tiplari qiymatlariga qarab 2 ta turga bo’lindi, Primitive va Non-primitive(Object).

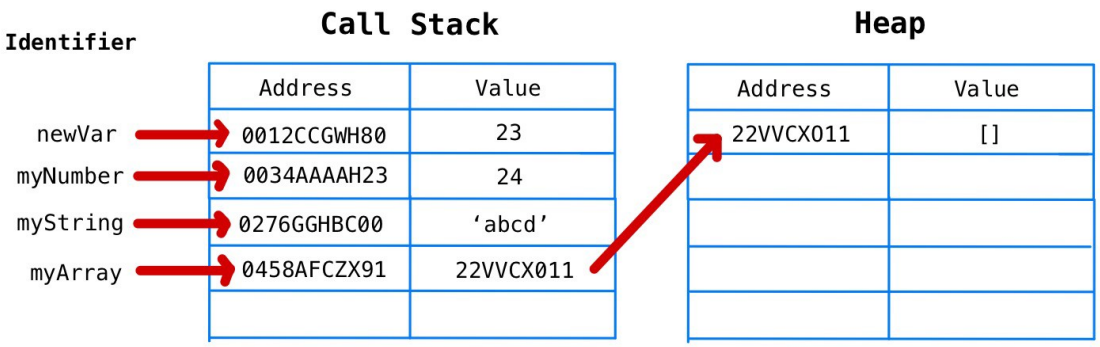
Primitive qiymatlarga:

1. String
2. Number
3. Boolean
4. Null
5. Undefined
6. BigInt

7. Symbol

▼ Obyektlar qanday ishlaydi ? Primitive qiymatlardan nima farqi bor ?

Obyektlar Primitive qiymatlardan farqli o'laroq, Call stack'da o'zida reference saqlaydi. U reference bizni u Obyektning Heap'dagi manzilida turgan qiymatiga olib boradi. Shu sabab Object'lar mutable, Primitive qiymatlar immutable deyiladi.



▼ Garbage collection vazifasi va maqsadi nima ?

Garbage collection - Heap to'lib qolmasligi uchun Garbage Collector keraksiz Obyektlarni xotiradan o'chirib turadi. Bu JS'da tayyor bor(*Boshqa ko'p tillarda dasturchi o'zi qiladi*). Shu holatni Garbage collection.

▼ this kalit so'zi nima ?

this - Qayerda qanday ishlatilishiga qarab quyidagicha qiymatlarga ega bo'ladi:

1. Funksiyalarda this = global object (strict mode'da undefined, function constructor'da shu funksiya orqali yasaladigan object'ga teng)
2. Obyekt metodlarida this = shu obyektga teng
3. Event ichida this = shu event'ni qabul qilayotgan elementga teng
4. Boshqa holatlarda o'zini ishlatsak this = global object

▼ call, apply va bind nima ?

Funksiya va metodlarda context(this)ni o’zgartirish uchun ishlatiladi. Bir-biridan farqlari quyidagilar:

1. call(newContext, arg1?, arg2?) - ishlatishimiz bilan funksiya/metodni yangi context bilan ishlatib yuboradi.
2. apply(newContext, [arg1?, arg2?]) - call bilan birxil, faqat funksiya/metodning argumentlarini array ko’rinishida bersak bo’ladi.
3. bind(newContext, arg1?, arg2?) - call bilan birxil, faqat funksiyani yangi context bilan qaytarib beradi. Qaytgan funksiya o’zgaruvchiga olinib, xohlagancha ishlatsak bo’ladi.

▼ Map, Set, WeakMap va WeakSet nima ?

Bular, JS’da mavjud bo’lgan data strukturalar. Tagi obyekt bo’lib, har birini o’z xususiyatlari bor. Masalan:

Map - Obyektlar bilan deyarli bir xil. Bitta farqi, Map bizga “key” uchun xohlagan tipdagi qiymatni berish imkonini beradi. Masalan bir obyektni key sifatida, ikkinchisini value sifatida.

Set - “Unique” qiymatlardan iborat “collection”. Ya’ni, har bir qiymat Set’ning ichida birmarta takrorlanadi. Set’da “key” yo’q.

WeakMap - Map bilan deyarli birhil. Ushbu farqlarini aytmasak: WeakMap “key” sifatida faqat obyekt oladi. WeakMap orqali iteration va WeakMap ichidagi qiymatlar va key’lar haqida ma’lumot beruvchi metodlardan foydalanib bo’lmaydi. Sababi esa, Garbage collection to’g’ri ishlashi uchun.

WeakSet - Set bilan birhil. Ammo faqat qiymat sifatida object oladi. Qolgan xususiyatlari WeakMap’dagidek.

▼ Type coercion va type conversion farqi nima ?

Type conversion - bir tipdan boshqa tipga o’girish. Bu JS tomonidan bevosita yoki siz ko’rsatgan buyruq orqali bo’lishi mumkin. (implicit or explicit conversion) Masalan, Number(”12”), 12 + “22”

Type coercion - JS tomonidan bevosita tipni boshqa tipga o’girish. (implicit conversion) 12 + “22”, "6" / "3”, alert({age: 2})

▼ Operatorlar turlari haqida gapiring.

Operatorlarni o’z vazifasidan kelib chiqib: Conditional, Comparison, Ariphmethic, Logical va boshqa operator turlari mavjud.

Shuningdek, ularni kattaroq sinflarga bo’lsa ham bo’ladi: Unary, Binary va Ternary.

Ba’zi mavzuga oid savollar:

1. == va === ning farqi nima ? 1 == !!“1”
2. &&, ?? va || farqi ?

▼ Oddiy va chiziqli funksiyalar farqi nima ?

Oddiy(regular) va chiziqli funksiya(Arrow function) farqlari quyidagilar:

1. Sintaksi harxil.
2. Arrow function’da “this” kalit so’zi yo’q. Shu sabab u o’zidan tepada bo’lgan “this”ni olib beradi.
3. Arrow function’da arguments obyekti yo’q.
4. Birhil nomli argumentlar berish mumkin emas. (*Oddiy funksiyada mumkin. Ammo “use strict”da unda ham mumkin emas*)
5. Arrow function’ni new kalit so’zi orqali Constructor sifatida ishlatib bo’lmaydi.

▼ Optional chaining nima ?

Optional chaining - biror obyektning biror property va method’lariga “*xavfsiz*” kirish usuli. Bu orqali kirilgan property yoki method agar yo’q bo’lsa bizga error emas, balki undefined qaytaradi.

▼ Tagged template literals (tagged strings) nima ?

Tagged template literals - bu template literal va funksiyalarni birlashtirgan holda template literal’larni parse qilib olishda ishlatiladi. Sintaksi quyidagicha:

```
// Syntax
tagFunction`template literal ${myVar}`;

// Example use case
function highlight(strings, ...values) {
  let str = "";
  strings.forEach((string, i) => {
    str += `${string} <span class='hl'>${values[i] || ""}</span>`;
```

```
  });
  return str;
}
const catName = "Tom";
const catAge = 15;
const sentence = highlight`My cat's name is ${catName} and he is ${catAge} years o
ld`;
document.body.innerHTML = sentence;
```

▼ Constructor functions nima ?

Constructor functions - bu birnechta birhil xossalarga ega bo'lgan obyektlarni yasashda ishlatiladigan funksiyalar. Ular orqali, birhil kod takrorlanishini kamaytirib, samarali kod yozamiz. Ular oddiy funksiyani oldiga new kalit so'zini qo'yish orqali yasaladi.

▼ Nechta Array metodlarini bilasiz ? Reduce va map farqi va ishlashi qanday ?

Reduce - ma'lum bir arrayni olib, undan yagona qiymat chiqarib olishda foydalaniladi. Bu natijaga, o'zining birinchi argumenti(accumulator) orqali erishadi. Ya'ni, accumulator butun loop jarayonida, biz qaytargan qiymat ta'siri orqali o'zgarib boradi va oxirgi qiymat sifatida qaytadi.

Map - ma'lum bir arrayni olib, uni ustida ishlab, yangi array qaytarishda ishlatiladi.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6b14289e-5959-4d90-a80d-6e1650470b15/JS_Array_Cheat_Sheet_-_Dark.pdf

▼ Destructuring assignment nima ?

Destructuring assignment - array va object'lar ichidagi ba'zi qiymatlarni "chiqarib olish" uchun ishlatilagan sintaks.

Misollar(*MDNdan olindi*):

```
const [a, b] = array;
const [a, , b] = array;
const [a = aDefault, b] = array;
const [a, b, ...rest] = array;
const [a, , b, ...rest] = array;
```

```
const [a, b, ...{ pop, push }] = array;
const [a, b, ...[c, d]] = array;

const { a, b } = obj;
const { a: a1, b: b1 } = obj;
const { a: a1 = aDefault, b = bDefault } = obj;
const { a, b, ...rest } = obj;
const { a: a1, b: b1, ...rest } = obj;
const { [key]: a } = obj;

let a, b, a1, b1, c, d, rest, pop, push;
[a, b] = array;
[a, , b] = array;
[a = aDefault, b] = array;
[a, b, ...rest] = array;
[a, , b, ...rest] = array;
[a, b, ...{ pop, push }] = array;
[a, b, ...[c, d]] = array;

({ a, b } = obj); // brackets are required
({ a: a1, b: b1 } = obj);
({ a: a1 = aDefault, b = bDefault } = obj);
({ a, b, ...rest } = obj);
({ a: a1, b: b1, ...rest } = obj);

let a = 1;
let b = 2;
[a, b] = [b, a];
```

▼ prototype nima __proto__ bilan farqi nimada ?

JavaScript’da barcha obyektlarda yashirin Prototype xossasi(*obyekt yoki null*ga *teng*) mavjud. Qaysidir obyekt ichida biz qidirayotgan xossa bo’lmasa, JS u xossani Prototype’dan qidiradi va topsa ishlatadi. Shuni Prototype inheritance deyiladi.

__proto__ bu getter/setter bo’lib, Prototype obyektini olishda yoki yangi qiymat berishda ishlatiladi. Ya’ni,

```
let obj = {};
obj.__proto__; // Prototype obyektini olib beradi.
obj__proto__ = {}; // obj'ning Prototype xossasi bo'sh obyektga teng bo'ldi.
```

Ammo, Prototype xossasini set qilishning boshqa yangiroq usullari bular, Object.create(newProto, [descriptors]?), Object.getPrototypeOf(obj), Object.setPrototypeOf(obj, proto)

▼ Property flags va descriptors nima ?

Property flags - bu object property'si ichidagi quyidagi yashirin attributlar(*flags*). (*default holatda true bo'ladi barchasi*)

1. writable - true bo'lsa, property qiymatini o'zgartirsa bo'ladi, false bo'lsa yo'q.
2. enumerable - true bo'lsa, shu property loop qilganda ko'rinadi, bo'lmasa yo'q.
3. configurable - true bo'lsa, property'ni o'chirsa bo'ladi, flag'larni o'zgartirsa bo'ladi, false bo'lsa yo'q. False qilingandan so'ng qayta true qilib bo'lmaydi.

Property descriptor degani esa, Object.getOwnPropertyDescriptor(obj, propName); metodidan qaytadigan qiymat. (tepadagi uchta flag'lar va qiymatlarini ko'rsatadi)

Property flag'larni o'zgartirish uchun Object.defineProperty(obj, propName, descriptor) yoki Object.defineProperties(obj, {…props}) dan foydalanamiz.

Read more

▼ Iterables nima degani ?

Iterables - qiymatlari for..of orqali olinsa bo'ladigan object'lar. Built-in iterable'larga Array, String, Map, Set'lar misol bo'la oladi.

O'zimizning iterable'larimizni [Symbol.iterator] metodini yozish orqali qilsak bo'ladi. Bu metod, ichida next() metodi yozilgan object qaytaradi va for..of aynan shu objectdan foydalanadi. next() metodi {done: true/false, value: "qiymat" } ko'rinishida object qaytarishi kerak. done true'ga teng bo'lsa, iteration tugaydi. Masalan:

```
const obj = {
    name: "Someone",
    age: 45,
    [Symbol.iterator]: function () {
        let index = 0;
        let values = Object.values(this);
        return {
              next() {
                  if (index < values.length) {
                      const value = values[index];
                      index++;
                      return { done: false, value: value };
                  }
                  else {
```

```
                    return { done: true, value: values[index] }
                }
            }
        }
    }
}
```

▼ Class nima ?

Class - obyektlar yasash uchun ishlatiladigan konstruktor funksiyaning bir ko'rinishi. Undan asosiy farqlaridan biri shuki, Class metodlarni o'z Prototype'iga joylaydi. Bu orqali, har safar obyekt yasalsa, shuncha ko'p metod emas, bitta metod hammasi uchun ishlaydi. Yana bir farqlaridan biri, Class'larda getter/setter'lar borligi. Bu orqali xossalarimizni xavfsizligini ta'minlaymiz. extends orqali esa bir Classni ikkinchisiga meros qilib bersak bo'ladi.(Inheritance)

Note: JSda Classlar syntactic sugar deyiladi, ya'ni allaqachon bor bo'lgan Constructor funksiyalarni osonlashtirish uchun ishlatiladi xolos. Boshqa ko'plab tillarda unday emas.

▼ JSda OOP va uning prinsipllarini bilasizmi ?

OOP nima - OOP bu dasturdagi kodlarni vazifasiga qarab, obyektlar orqali turli qismlarga ajratib, kodlarni takrorlanishlarini oldini olib, qayta ishlatish imkonini beradigan -PROGRAMMING PARADIGM.

Uning 4 ta asosiy prinsipllari mavjud, ular:

1. **Inheritance** - bir Class'ni ikkinchi Class'ga meros qoldirish, yoki bir Class'dan ikkinchi Class'da foydalanish. (kaliti so'zlar: extends, super)
2. **Encapsulation** - deb Class tashqariga o'zidan nimani ko'rsatishi va ko'rsatmasligiga aytiladi. Ya'ni, public, private va protected bo'lgan property va methodlar orqali qilinadigan amallar Encapsulation qilishga kiradi(*getter/setterlar ham*). JS'da barcha property va methodlar default holatda public bo'ladi. Shuningdek, bu prinsipl harbir Class o'z ishini qilishi kerak degan tushunchani targ'ib qiladi.
3. **Abstraction** - bu Class faqatgina ishlatilinadigan property/method'larni tashqariga chiqaradi, keraksizlarni o'z ichida qoldirishi(yashirish). Ya'ni, orqa fondagi ishlar orqa fonda qoladi.

4. **Polymorphism** - metodlar sharoitga qarab harxil qiymatlar qaytarishi. Masalan, .sayName() metodi, harbir User'ning ismini chiqaradi, aynan bittasini emas. Metod qayerda ishlatilsa shuyerga qarab to'g'ri ishlaydi. Polymorphism'ni ikki qismga bo'lsak bo'ladi, 1. Paramater-based. (Constructor'dan kelgan qiymatlar) 2. Inheritance-based. (Metodlarni ustidan yozish)

▼ Functional Programming nima ?

Functional programming - declarative programming paradigm'lar sinfiga mansub bo'lib, dasturni pure funksiyalarga bo'lish orqali tuzishni ilgari suradi. Bu orqali kod tushunarli, boshqarishga oson, qayta-ishlatsa bo'ladigan va testlashga oson bo'ladi. Functional Programmingning quyidagi konseptlari mavjud: Immutability, Pure functions, Higher order functions, First-class functions, Currying, Recursion, Composition, Referential Transparency va hkz.

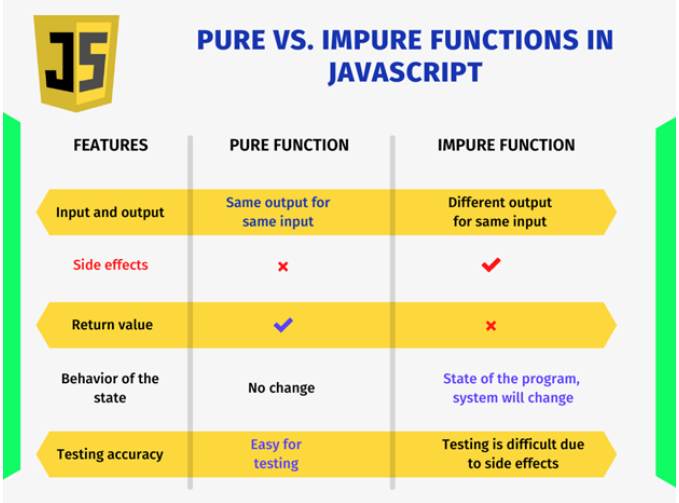
▼ Immutability tushunchasi nima ?

Immutability - bu dasturdagi data(object, array…)larni aslini o'zgartirmasdan ular bilan ishlash. Bunda Arraylar bilan ishlaganda, mutable methodlardan iloji boricha qochiladi. Masalan,

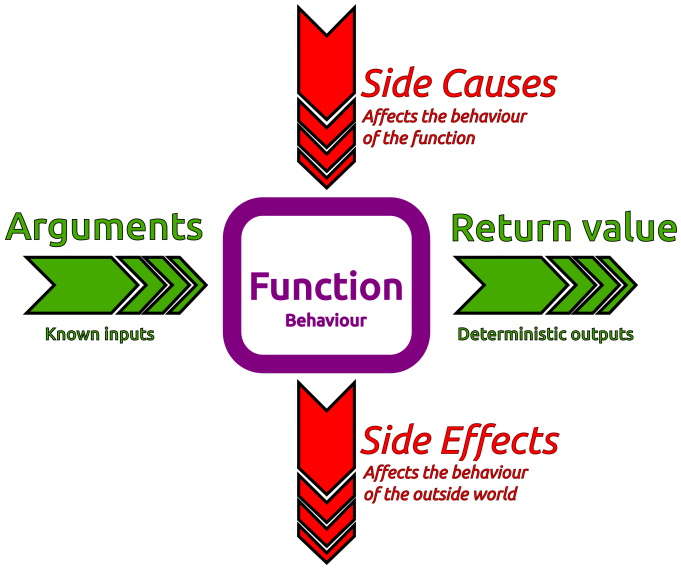
```
const numbers = [1,2,3,4];
const doubledNums = doubleNums(numbers);

function doubleNums(nums) {
  return nums.map(num => num + num);
}
```

▼ Pure va impure funksiya nima ?

Pure functions - qaytaradigan natijasi side effect'lardan xoli va har safar birhil inputga birhil output qaytaradigan funksiyalar. Impure esa aksi.



Side effect - funksiya tashqarisidan nimanidir ishlatish. Tashqari bilan to'g'ridan to'g'ri ishlash.



▼ First-class functions nima ?

First class functions(citizens) - bu variable'ga bersa bo'ladigan, argument sifatida bersa bo'ladigan, boshqa funksiyadan return qilsa bo'ladigan va property berilsa saqlay oladigan funksiyalar. (Funksiya obyektlardek ishlay olishi)

▼ Higher-order Functions nima ?

Higher order functions - bu argument sifatida funksiya oladigan yoki qaytaradagan funksiyalar. Bularga misol, Array.prototype.map,

Array.prototype.reduce …

▼ Currying nima ?

Currying - funksiyalarni birnechta qismlarga bo'lib, alohida alohida, argument berib, chaqirish usuli. Misol:

Simple function: add(1, 2, 3); // 6

Curried function: add(1)(2)(3); // 6

▼ Recursion nima ?

Recursion - Functional programming'da bir funksiya o'zini o'zi chaqirishiga nisbatan aytiladi va bunday funksiyalar "recursive functions" deyiladi. Masalan, ushbu 1 dan N songacha bo'lgan sonlar ko'paytmasini topish uchun Recursion'dan foydalanib ko'ramiz. *P.s: Recursive funksiyalar dasturni ishlashiga yomon ta'sir qilishini inobatga olib, hamma joyda ham ishlatish maslahat berilmaydi.*

```
function multiplyUpTo(num) {
    if(num === 1) return 1;
    return multiplyUpTo(num - 1) * num;
}
multiplyUpTo(4); // 24
// 3 * 2 * 1
// 1 * 2 * 3 * 4 = 24
```

▼ Callback function nima ?

Calback function - argument sifatida boshqa funksiyaga berilib, shu funksiya ichida chaqiriladigan funksiyalar - callback funksiyalar deyiladi.

▼ Function Composition nima ?

Function composition - deb bir data ustida natijaga yetguncha, funksiyalar ketma-ketlikda Callback va Currying uslubidan foydalanib ishlashiga aytiladi.

Misollar:

```
// Osh uzatish :D
const compose = (fn1, fn2) => osh => fn1(fn2(osh));

const addMeat = (osh) => osh + " + Meat";
const addEgg = (osh) => osh + " + Egg";

const osh = compose(addEgg, addMeat);

console.log(osh("Osh"));



// 2-misol, Eng ko'p ishlatilgan belgini(harf...) topish.
const compose = (a, b) => str => a(b(str));

const countChars = (str) => {
    const obj = {};
    for (const char of str) {
        obj[char] = obj[char] + 1 || 1;
    }
    return obj;
};

const findMaxLetter = (obj) => {
    let max = 0;
    let result = { count: 0, letter: '' };

    for (let key in obj) {
        if (obj[key] > max) {
```

```
            max = obj[key];
            result.count = obj[key];
            result.letter = key;
        }
    }
    return result;
}

const getMaxLetter = compose(findMaxLetter, countChars);

console.log(getMaxLetter("Some value!")); // {count: 2, letter: 'e'}
```

▼ Pipe Function nima ?

Pipe Function - Function Composition bilan deyarli birhil, faqat ustun tarafi Pipe function'ga berilgan funksiyalar ketma-ketlikda ishlaydi.

```
const pipe = (...fns) => (data) => fns.reduce((data, fn) => fn(data), data);

const addMeat = (osh) => osh + " + Meat";
const addEgg = (osh) => osh + " + Egg";

const osh = pipe(addMeat, addEgg, addMeat, addEgg);
console.log(osh("Osh")); // Osh + Meat + Egg + Meat + Egg
```

▼ Referential Transparency nima ?

Referential Transparency - biror qiymat qaytaradigan funksiyani argument sifatida boshqa funksiyaga bergan holatdagi natija, shu funksiya qaytaradigan qiymatni argument sifatida bergandagi bilan birhil bo'lishi kerak, shunda shu birinchi funksiya "Referential transparent" deyiladi. Ya'ni,

```
// Correct
const sum = (a, b) => a + b;
const showSum = (num) => "Result is: " + num;

showSum(sum(1, 2)); // Result is: 3
showSum(3); // Result is: 3


// ------------------------------------- //
// Incorrect
const sum = (a, b) => {
  console.log(a + b);
  return a + b;
};
```

```
const showSum = (num) => "Result is: " + num;

showSum(sum(1, 2));
showSum(3);
```

▼ IIFE nima ?

IIFE(*Immediately Invoked Function Expression*) - e’lon qilingan zahoti ishlaydigan funksiya. Ko’proq global scope’da kod yozishdan qochish uchun ishlatiladi.

Masalan,

```
(function() {
   console.log("I am in a scope!");
 })()
```

▼ Execution context haqida gapiring.

Har bir block kod uchun JS tomonidan ochiladigan ma’lumotlar bloki bo’lib, ayni damda ishlayotgan kod uchun kerak bo’lgan barcha ma’lumotlarni o’zida jamlaydi. Masalan, this kalit so’zi, o’zgaruvchilar/funksiyalar…

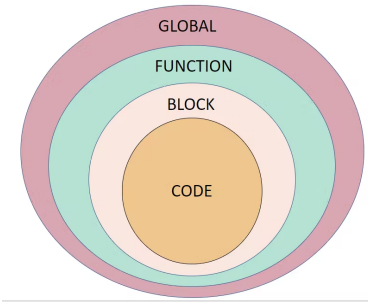
(EC)Execution context ikki turda bo’ladi, Global va Function execution context.

Harbir EC ikki bosqich(phase)ga bo’linadi.

1. Creation phase (Yasalish bosqichi) Bu bosqichda quyidagilar ro’y beradi:

   1.1 O’zgaruvchi va funksiyalar declaration’larini saqlash.

   1.2 Scope chain yasalishi.



   1.3 this kalit so’ziga qiymat berish.

2. Execution phase (Kodni ishlatish bosqichi, o’zgaruvchilarga qiymat berish…)

▼ Lexical environment nima ?

Lexical environment - ayni damda ishlab turgan Execution context'da ishlatilgan variable/function'lar saqlanadigan joy. Kod qayerdaligiga qarab ichma-ich kirib ketgan bo'ladi. Masalan,

```
/*
  Global Execution Context
  Lexical environment = {
       environmentRecord: { x: 12 },
       outer: null
 }
*/
let x = 12;

function myFn() {
  /* Lexical environment = {
       environmentRecord: { n: 30 },
       outer: [Tepadagi lexical environment]
 } */
  let a = 30;
  alert(y);
  function innerFunction() {
    /* Lexical environment = {
       environmentRecord: { a: 50 },
       outer: [Tepadagi lexical environment]
     } */
      let a = 50;
      alert(x);
  }
}
```

▼ Scope nima ?

Scope - o'zgaruvchi va funksiyalar qayerda ko'rinish/ko'rinmasligini aniqlab beruvchi tushuncha. (Global, local, block scope)

▼ Closure nima ?

Closure - deb funksiyaning o'zining Scope chain'idagi o'zgaruvchilarni ishlatishga ruhsati bo'lishiga aytiladi, xattoki agar u ichki funksiya bo'lib, tepadagi funksiya ishlab tugagan bo'lsa ham.

▼ JSda modullar qanday ishlaydi ?

Modullar katta JavaScript dastur kodlarini import/export orqali qismlarga bo'lib ishlatish. Bunda harbir modul alohida faylda bo'ladi. Har bir alohida fayl, o'zidan

funksiya, class, o’zgaruvchi va hokazolarni tashqariga chiqaradi yoki tashqaridan olib ishlatadi. Bu orqali kodimiz tushunarli, qayta ishlatishga oson va yaxshi strukturaga ega bo’ladi.

Oddiy JS scriptlaridan modul JS scriptlarini farqi,

1. Modul fayllar avtomatik ravishda “use strict” ishlatadi.
2. Har bir modulda alohida scope mavjud.
3. type=”module” bo’lgan fayllar avtomatik “defer” atributini ham ishlatadi. (Shunday ishlaydi)
4. this kalit so’zi undefined.

More: modules , import/export, dynamic imports

▼ Generators nima ?

Generators - har chaqirganda ketma-ketlikda iterable object sifatida qiymat qaytaradigan funksiyalar. Masalan,

```
function* generateNumbers() {
  yield 1;
  yield 2;
  return 3;
}

// Generator funksiya iterable "generator object" qaytaradi.
let generator = generateNumbers();

generator.next(); // { value: 1, done: false }
generator.next(); // { value: 2, done: false }
generator.next(); // { value: 3, done: true }
```

▼ Asinxron JavaScript (Callbacks, Promises, async/await) haqida gapiring.

Asinxron dasturlash deb kodni dasturni ishlashini bloklamaydigan holda yozish va asinxron kodlar bilan to’g’ri ishlashga aytiladi. Asinxron kodlarga misol, settimeOut, setInterval, Event Listener, Http Request’lar misol bo’ladi.

Ular bilan ishlash uchun, Callback, Promise, async/await lardan foydalanamiz.

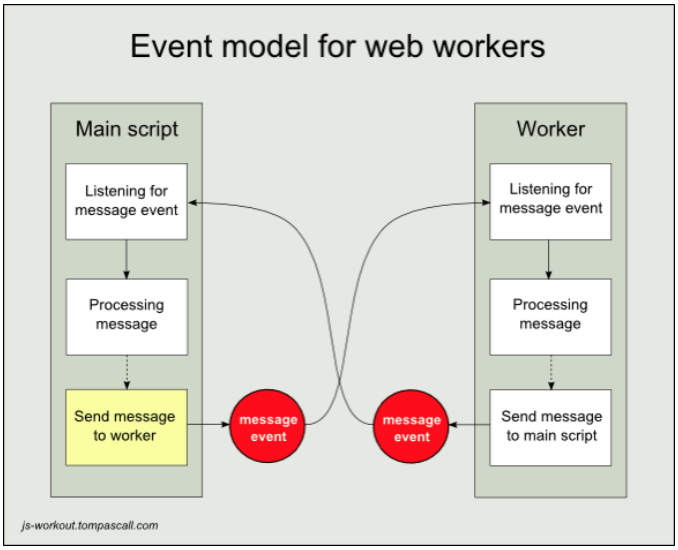
```
// --------- Callbacks ---------

setTimeout(() => {
    console.log('Coming from Callback');
},2000);

function getData(callback) {
    const Http = new XMLHttpRequest();
    const url='https://jsonplaceholder.typicode.com/posts';
    Http.open("GET", url);
    Http.send();

    Http.onreadystatechange = (e) => {
      callback(e);
   }
}
getData((e) => {})

btn.addEventListener('click', () => {
  console.log('button clicked');
})

// --------- Promises ---------
fetch(url)
   .then(data => data.json())
   .then(res => console.log(res));

const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Coming from promise");
  }, 2000);
});

myPromise
  .then(data => data + ", yes it is right!")
  .then(data => data + ", surely")
  .then(console.log);

// Async await
async function someFunction() {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("From async/await!"), 2000);
  });

  let result = await promise; // promise'ni kutib turadi.

  console.log(result); // "From async/await!"
}

someFunction();
```

More: https://javascript.info/promise-basics

▼ Web workers nima va nima ustunligi bor ?

Web workers - asosiy thread'dan tashqarida, yangi faylda yangi thread'da katta hajmli kodlarni ishlatib, javobini asosiy threadga qaytarishda ishlatiladigan Web API.

Ishlashi quyidagicha,

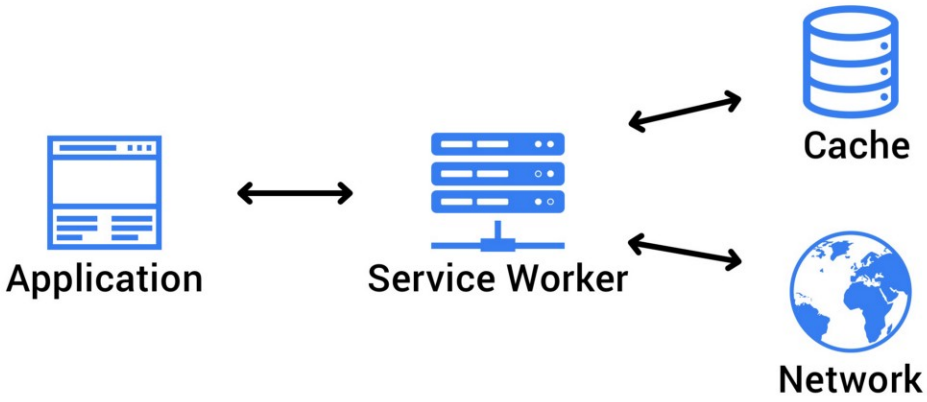


Misol,

```
// main.js
let worker = new Worker('worker.js');

worker.onmessage = function (event) {
  console.log("Result", event);
};

// worker.js
let result = 0;
for (let i = 0; i <= 1000000000; i++) {
   result += i;
}
postMessage(result);
```

▼ Service workers nima va nima ustunligi bor ?

Service workers - browser, dastur va internet orasida “ma’lum bir“ maqsadlarda ishlatsa bo’ladigan Web API**.**



▼ JSda HTTP so'rovlarni qanday yuborish mumkin ?

HTTP so’rovlarni asosan, quyidagi 3 ko’rinishda qilish mumkin.

▼ XHR(*XmlHttpRequest*)

```
// XHR
const postsAPI = 'https://jsonplaceholder.typicode.com/posts';

function sendRequest(method, url, body) {
  return new Promise((resolve, reject) => {

    const xhr = new XMLHttpRequest();
    xhr.open(method, url);
    xhr.responseType = 'json';

    if(data) xhr.setRequestHeader("Content-type", "application/json");

    xhr.onload = () => {
      if (xhr.status >= 400) reject(xhr);
      else resolve(xhr.response)
    }
    xhr.onerror = () => reject("Error!");

    xhr.send(JSON.stringify(body));
  })
}
```

```
document.getElementById('button').addEventListener('click', () => {
  sendRequest("POST", postsAPI, {}).then(console.log).catch(console.log);
})
```

▼ Fetch

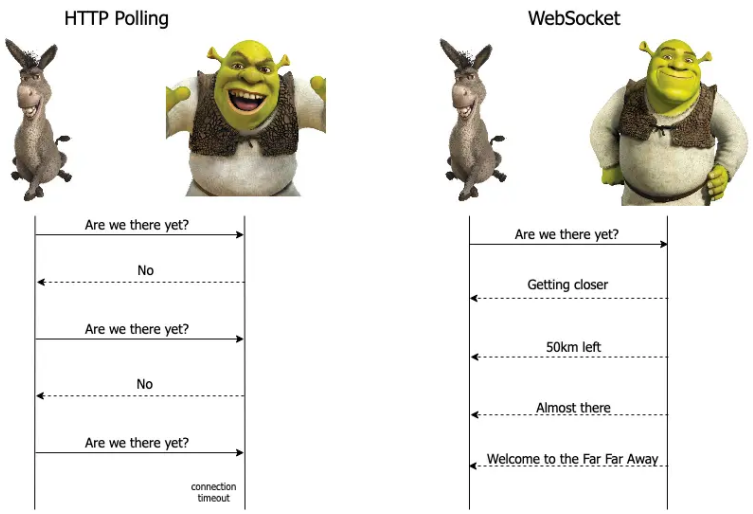
```
// Fetch
const postsAPI = 'https://jsonplaceholder.typicode.com/posts';

function sendRequest(method, url, body) {
  return fetch(url,
    {
      method: method,
      body: JSON.stringify(body),
      headers: body ? { 'Content-type': 'application/json' } : {}
    })
    .then(response => {
      if (response.status >= 400) {
        throw Error("Error occured");
      }
      return response.json();
    })
};

document.getElementById('button').addEventListener('click', () => {
  sendRequest("POST", postsAPI, { someData: "Data" })
    .then(console.log)
    .catch(console.log);
})
```

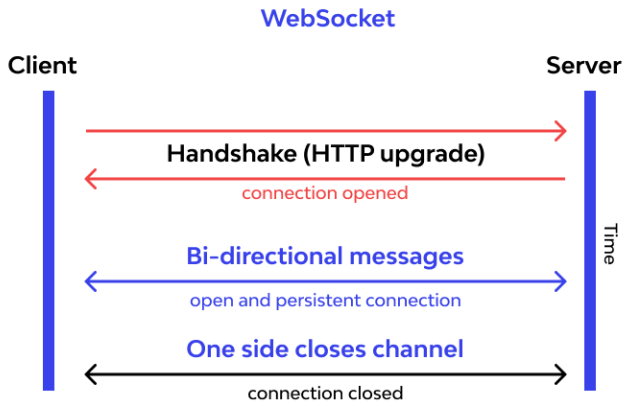
▼ Axios

https://axios-http.com/docs/intro

▼ WebSocket nima va nimalarga foydalaniladi ?

Websocket - client va server o’rtasida ikki tomonlama aloqalarni o’rnatuvchi texnologiya. HTTP’dan farqli jihati, HTTP bitta so’rov bilan bitta javob oladi va shu bilan aloqa uziladi. WebSocket’larda esa, bitta so’rov(*Handshake* 🤝) bo’ladi va bir tomon aloqani uzmaguncha aloqa saqlanib qoladi, orada xohlagancha ma’lumot alishish ro’y berishi mumkin.
WebSocket’lar onlayn chat yasashda ideal yo’l hisoblanadi. WebSocket’lar qurishda ko’proq socket.io kutubxonasi ishlatilinadi.



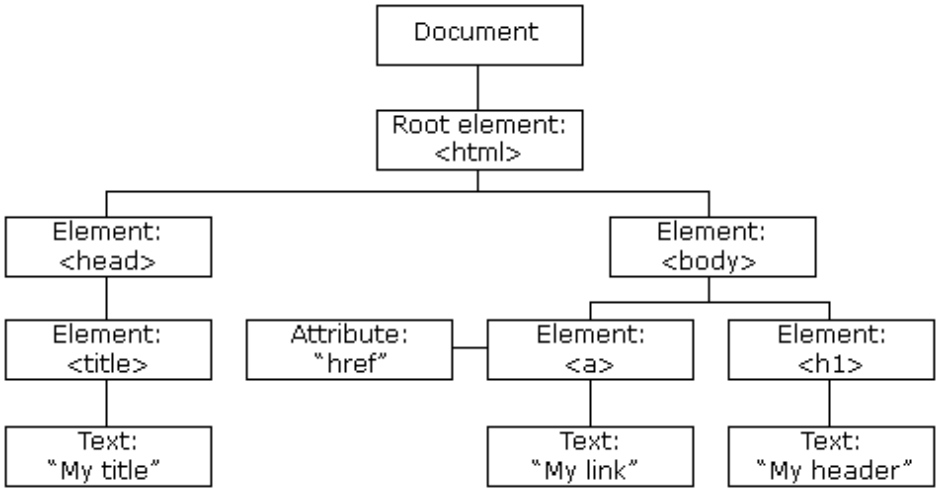


Demo: https://socket.io/demos/chat/

▼ DOM

▼ DOM nima ?

DOM - Document object model.



▼ Elementlarni qanday yo'l bilan tanlab olishlarni bilasiz ?

- `getElementById` – search element by element_id
- `getElementsByTagName` – search element by tag name (e.g., span, div)
- `getElementsByClassName` – search element by class name
- `getElementsByName` – search element by name attribute
- `querySelector` – returns the first element that matches the specified selector
- `querySelectorAll` – returns elements that match the specified selector

▼ HTMLCollection va NodeList farqi nima ?

HTMLCollection bu document element'lardan tashkil topgan kolleksiya.

NodeList esa document nodes (element nodes, attribute nodes, and text nodes)lardan tashkil topgan kolleksiya.

▼ innerHTML va textContent farqlari nima ?

• textContent – gets and sets the text content of a node.

• innerHTML – gets and sets the HTML content of an element.

▼ append va prepend ning farqi nima ?

- append() – inserts a node after the last child node of a parent node.
- prepend() – inserts a node before the first child node of a parent node.

▼ Qanday qilib birnechta elementlarni qo’shishimiz mumkin. Masalan 100ta p tegi.

DocumentFragment() orqali. Quyidagi kod bunga misol:

```
let languages = ['JS', 'TypeScript', 'Elm', 'Dart','Scala'];
let langEl = document.querySelector('#language')

let fragment = new DocumentFragment();

languages.forEach((language) => {
  let li = document.createElement('li');
  li.innerHTML = language;
  fragment.appendChild(li);
})

langEl.appendChild(fragment);
```

▼ Qanday qilib elementga yangi class qo’shish yoki o’chirishimiz mumkin ?

element.classList orqali. Masalan:

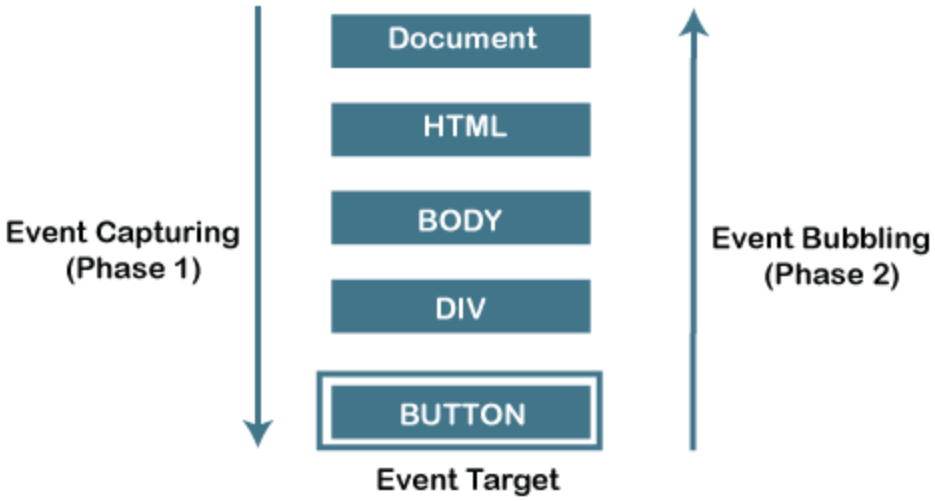
element.classList.remove('myClass') - removes “myClass” from the classlist.

element.classList.add('myClass') - adds “myClass” to the classlist.

▼ event bubbling va capturing nima ?

Event bubbling - is the event starts from the target element to the top element.

Event capturing - is the event starts from top element to the target element. It is the opposite of Event bubbling, which starts from target element to the top element.



▼ Event delegation nima ?

Event Delegation is basically **a pattern to handle events efficiently** . Instead of adding an event listener to each and every similar element, we can add an event listener to a parent element and call an event on a particular target using the . target property of the event object.

▼ Bu ikki kodning farqi nima ?

```
btn.onclick = function() {
alert(this.id);
};

//-------------------//

btn.addEventListener("click", function() {
  alert(this.id);
});
```

▼ BOM

▼ BOM nima ?

📌 BOM - bu “Browser object model”. Bizga browser bilan aloqa qilishga yordam beradi. BOM quyidagi 5 ta obyektlarni o’z ichiga oladi.

▼ Window

- window object is the global object.
- The window object exposes the browser's functionality. Like:
  1. Window size (innerWidth, innerHeight, outerWidth, outerHeight)
  2. Open (close) a new window (window.open(url, windowName, [windowFeatures]))
  3. Resize a window (window.resizeTo(width, height), window.resizeBy(deltaX,deltaY))
  4. Move a window (window.moveTo(x, y))
  5. Window opener (window.opener)

▼ Location

- The Location object represents the current location (URL) of a document.
  - The Location object contains these properties and methods.
    1. location.href (example: "https://joshdeveloper.com/blog/videos?title=BOM")
    2. location.protocol ("http:" , "https:")
    3. location.host (joshdeveloper.com)
    4. location.pathname (/blog/videos)
    5. location.search (?title=BOM)
    6. location.reload(true?)
    7. location.assign('url')
- We can retrieve query values(for example "title=BOM&author=JoshDeveloper") using location.search and new UrlSearchParams().

```
// location.search = title=BOM&author=JoshDeveloper

const urlParams = new URLSearchParams(location.search);
for (const [key, value] of urlParams) {
console.log(${key}:${value});
}

// Output: title:BOM  author:JoshDeveloper
```

▼ Navigator

Foydalanuvchi qurilmasi va browser'i haqidagi ma'lumotlarni olish imkonini beradi. Masalan, navigator.clipboard, navigator.bluetooth, navigitor.geoLocation(), navigator.getBattery() va hkz.

▼ Screen

Foydalanuvchi ekrani(Screen) haqida ma'lumotlarni olsak bo'ladi. Masalan,

- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

▼ History

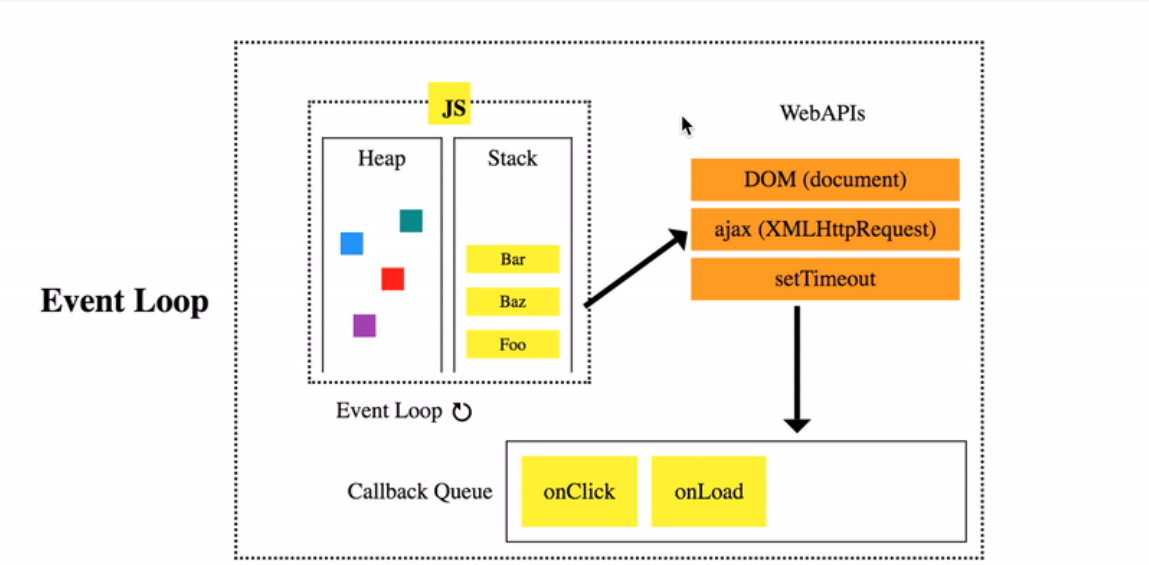
Bu orqali Browser tarixi bilan ishlay olamiz. Ya'ni, oldingi ochilgan sahifalarga qayta kira olamiz. Quyidagi metodlar orqali: history.back(), history.forward()

▼ Storages

Features	Local Storage	Session Storage	IndexedDB	Cookies	Web SQL
Data size	5MB	5-10MB	User disk space	4KB	User disk space
Supported data type	Strings	Strings	Any JS datatype	Strings	SQL data types
Query Processing speed	200-300ms	200-300ms	300-350ms	200-300ms(Depends on API calls)	750-850ms
Lifetime	Till deleted	Till tab close	Till deleted	As defined	Till deleted
Browser support	High	High	High	High	Low
Secure data storage	no	no	no	no	no
Editable by users	yes	yes	yes	yes	yes

▼ Event loop haqida gapiring.

Event loop - bu JavaScriptda dastur davomida ishlashi kerak bo'lgan kodlarni Call Stack, Callback va Job Queue'ni kuzatib turish orqali, ishlatib beradi. Callback queue'ga tushgan funksiyalar Macrotasks deyiladi. Promise.then()'lar esa Microtasks deyiladi.

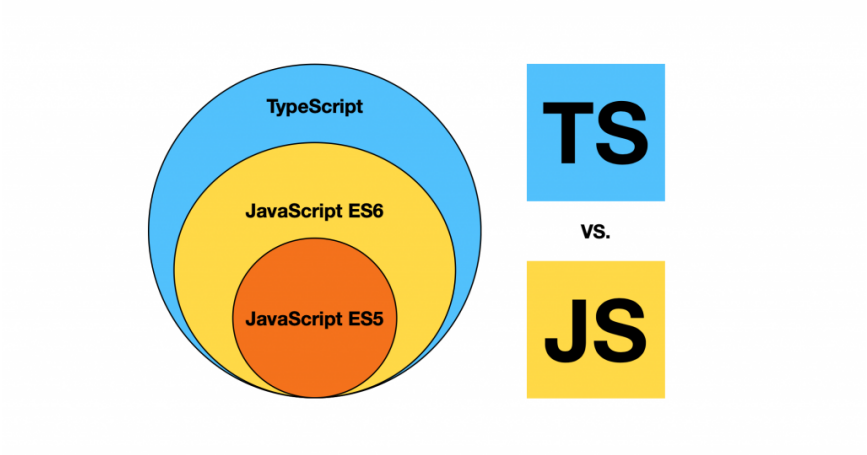


▼ TS

▼ TS va JS farqi

Parameter	JavaScript	TypeScript
Definition	JavaScript is a scripting language with first-class functions to create dynamic web pages.	TypeScript is a powerful object-oriented language as a superset to JavaScript, with generic and JS features to overcome the complexities of JS.
Typing	Weakly typed. In JavaScript, only dynamic typing is supported, not static typing.	Strongly Typed. TypeScript supports both static and dynamic typing.
Compilation	JavaScript does not need a compilation.	TypeScript needs to be compiled to JavaScript.
Error Detection	In JavaScript, errors are detected only at the run time.	Errors are detected or highlighted in the early development stage in TypeScript or in compiling stage.

Generics	JavaScript does not support generic features.	TypeScript supports generic features. It allows the creation of reusable components.
Execution	JavaScript runs directly on the browser and supports cross-platform, cross browsers.	TypeScript does not run directly on the browser.



▼ TypeScriptda mavjud oddiy tiplar

any, unknown, arrays, objects, union, literal, tuples, function types, void, never, enums

▼ Type aliases nima ?

Creating new name for specific type(s).

type MyType = string | number;

▼ Class va Interface'lar va ularning yangi xususiyatlari

Interface - biror object yoki class amal qilishi kerak bo'lgan struktura ko'rinishi.

(Class) Access modifiers(private, public, protected and readonly), implements interfaces, abstract classes

▼ Type casting nima ?

Type casting - bu xuddi type conversion'dek, bir tipdan boshqa tipga o'girish.

▼ Function overloads nima ?

Funksiyalarni harxil tipli ko’rinishda ishlashi uchun yordam beradi. Generic’ga o’xshash, ammo faqat biz bergan tiplargina mumkin bo’ladi. Masalan:

```
function add(a:string, b:string):string;

function add(a:number, b:number): number;

function add(a: any, b: any): any {
    return a + b;
}
```

▼ Generics nima ?

slider

▼ Utility types nima ?

Tiplar ustida ishlashga qo’shimcha yordam beruvchi tiplar. Bunga misol, Partial<type>, Required<type>, Readonly<type>, Pick<type>, Omit<type>

Batafsil: https://www.typescriptlang.org/docs/handbook/utility-types.html

▼ Decorators nima ?

Decorators - Class, property, paramater, method’lar ustida qo’shimcha ish bajarish ustida ishlatiladi.

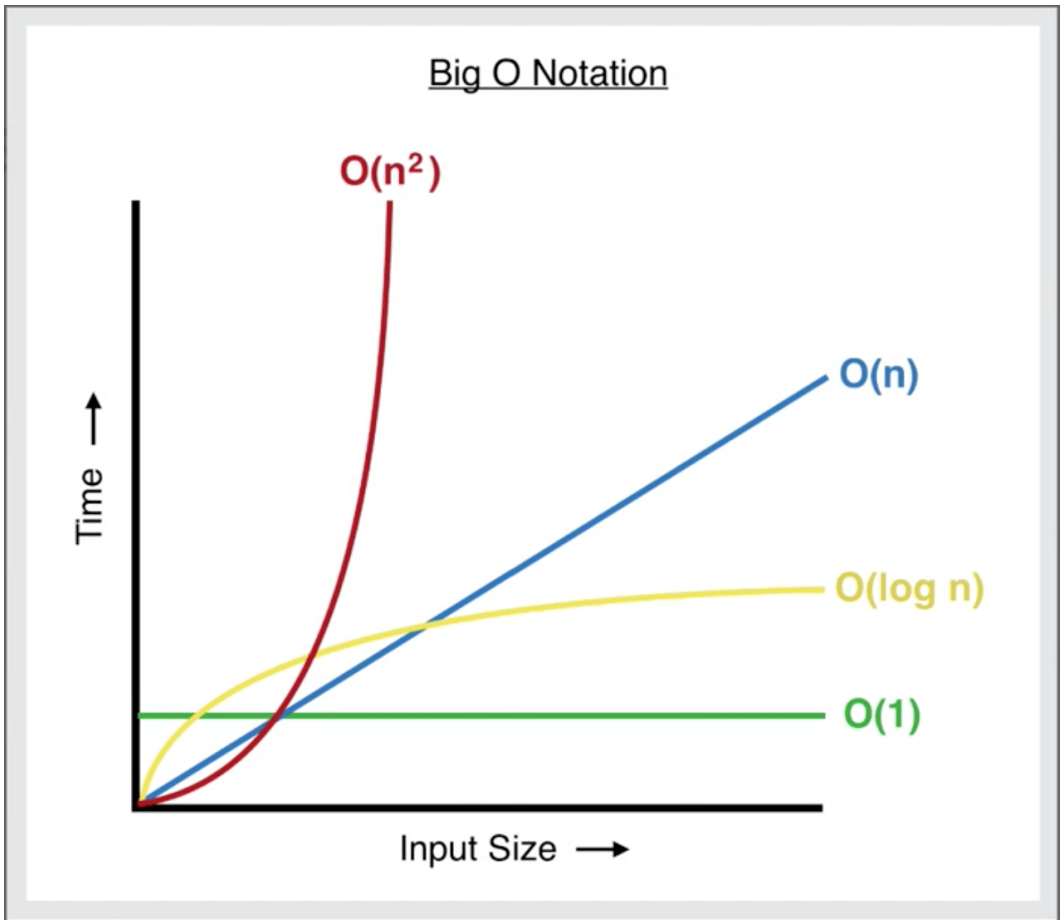
▼ TypeScript best practices haqida gapiring.

1. any tipidan kam foydalaning.
2. Class’larda access modifier’larni to’g’ri ishlating.(public, private va protected)
3. Function overload va Generic type’larni to’g’ri joyda ishlating.
4. Tsconfig’da strict check’larni o’chirmang.
5. Type aliases va Interface’lardan to’g’ri foydalaning.
6. Funksiyalarning return tipini to’g’ri bering.

▼ Umumiy Frontend

▼ Time va Space complexity nima ? (Algorithmlarga oid kichik bo’lim)

Algoritmlarni vaqti va qancha joy olishini o’lchash uchun biz Big O Notation’dan foydalanamiz.



Manbalar:
https://youtu.be/WqrbIUggEXQ (O’zbekcha)

https://www.udemy.com/course/js-algorithms-and-data-structures-masterclass (English)

▼ Dasturda muommo chiqsa qanday debug qilasiz ?

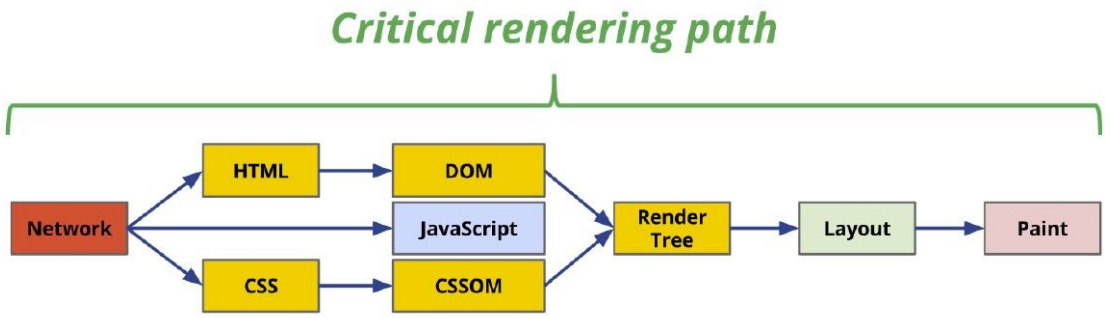
1. Console
2. Debugger
3. Tests

▼ Dasturingizga Freymvork va kutubxona tanlashda nimalarga e’tibor berasiz ?

1. Dasturni qanchalik katta ekanligi va nimalar talab qilishi
2. Freymvork tanlanganda, uni tezligini dasturga mosligi
3. Freymvork dokumentatsiyasi yaxshiligi va jamiyati katta kichikligi
4. Jamoa
5. Kelajagi(+Support)
6. Tanlangan boshqa texnologiyalar bilan “chiqishishi”

▼ Critical rendering path nima ?

CRP - bu browser sahifani to’liq yuklab, bizga ko’rsatguncha bo’ladigan jarayon. Quyidagi qadamlarni o’z ichiga oladi: DOM, CSSOM, Render Tree, Layout va Paint.



▼ Frontendda optimizatsiya qilish texnikalari haqida gapiring.

1. “Responsive“ rasmlar va WebP formatidagi rasmlardan foydalanish
2. Lazy loading
3. Minifying
4. Compressing(Gzip va Brotli)
5. Prefetching
6. CSS fayllarni qismlarga bo’lish, asinxron yuklash
7. JavaScript fayllarini asinxron yuklash

8. Tree shaking

▼ Tree shaking nima ?

Ishlatilmagan kodlarni build vaqtida tushirib qoldirishga Tree shaking deyiladi.

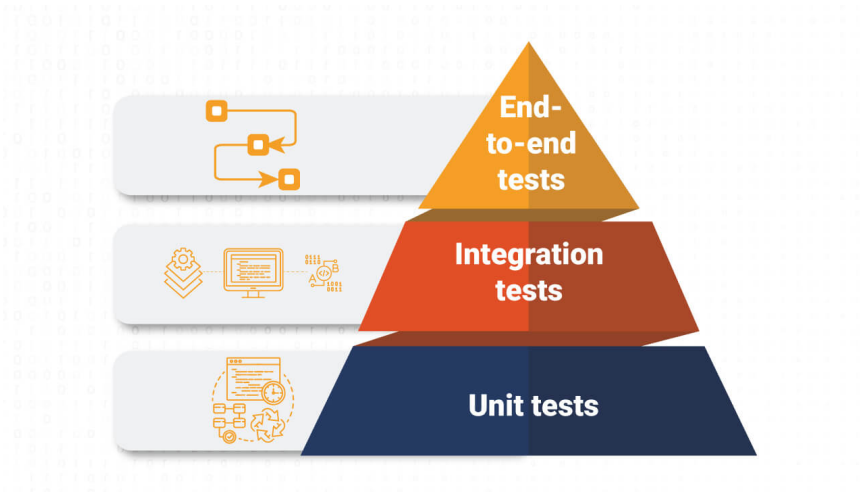
▼ Linting nima ?

Linting - linter orqali koddagi xatoliklarni aniqlab beruvchi “tool”.

▼ Linter va prettierni avtomatlashtirishni bilasizmi ?

https://gist.github.com/estorgio/e8bcaa8e87d0fcdcf85fdf598956e34c

▼ Testing turlari va foydalari ?



https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

▼ CORS nima ?

CORS (Cross-Origin-Resource-Sharing) bu - Ikki xil manzildan turib, ma’lumot alishish mexanizmi. Browser default holatda, bir website o’zini ichida ma’lumot alishishigagina ruhsat beradi(Same-origin-Policy). Agar boshqa URL’dan ma’lumot olmoqchi/yubormoqchi bo’lsangiz va u tomon sizga “ruhsat” bermagan bo’lsa, CORS xatoligini olasiz.

▼ SPA nima va nima ustunlik va kamchiliklari bor ?

SPA - Single Page Application.

Pros

1. Dastur tezligi
2. Osonroq PWA yasash

Cons

1. SEO bilan ishlash qiyin
2. Og’ir “first-time-load”
3. Eski browser’lar “support” qilmaydi

▼ SSR nima va nima ustunlik va kamchiliklari bor ?

SSR - Server Side Rendering.

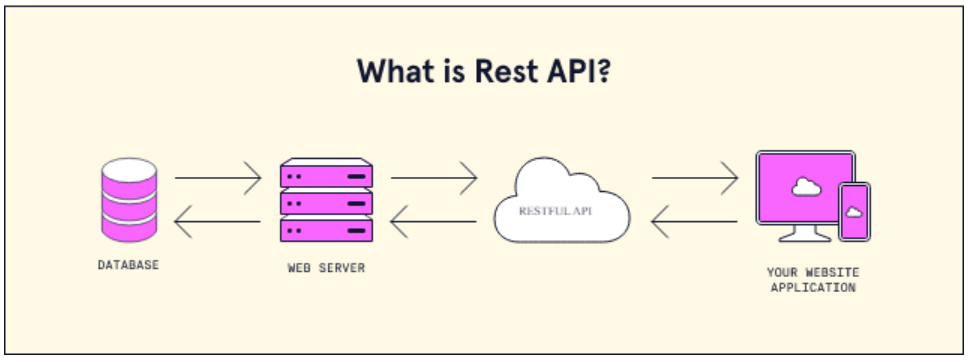
Odatiy Client Side Rendering’dan farqli o’laroq, SSR sahifalarni Serverda static yoki dynamic ravishda render qiladi va tayyor string ko’rinishidagi sahifani Client’ga yuboradi. Bu orqali saytda tezlik va SEO juda kuchayadi.

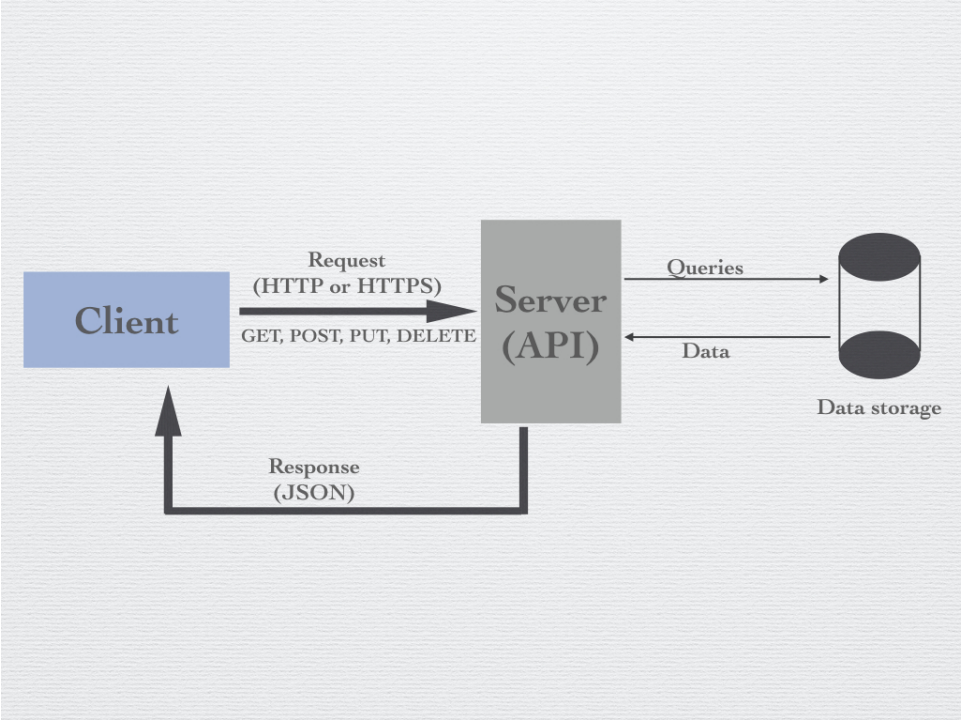
▼ PWA nima ?

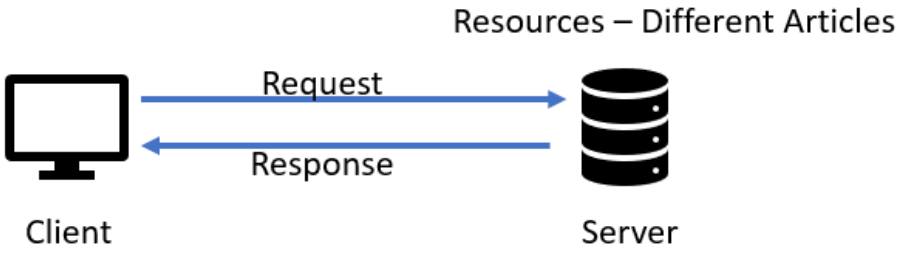
https://blog.logrocket.com/building-pwa-react/

▼ RESTful API

Client va Server o’rtasida turuvchi ma’lumot almashish uchun ishlatiladigan interface.







Endpoint	www.medium.com
Method	GET, PUT, POST, DELETE, PATCH
Header	Authentication information for both client and server
Body	Any information to be sent to server

▼ Webpack nima ?

Webpack - “module bundler” hisoblanadi va katta dasturlardagi JS, CSS kabi fayllarni bir faylga “bundle” qilishda ishlatiladi.

Batafsil: webpack.js.org

▼ CI/CD nima ?

CI- Continious Integration. CD - Continious Delivery/Deployment.

Biror loyihani yozishda, ba’zi inson qo’li bilan qilinadigan ko’p ishlarni avtomatlashtirish. Masalan, davomiy har bir yangi kodni testlab borish, harxil turdagi tekshiruvlar va statistikalar olib borish, deploy qilish kabi ishlar. Bunda bizga quyidagi texnologiyalar yordam beradi.

CircleCI, TeamCity, Jenkins, Github Actions, Gitlab CI/CD, Travis CI, Bamboo … va hkz

https://github.com/josh-developer/CI-CD-practice

▼ SEO nima va qanday optimizatsiya qilinadi ?

1. Ichki kontentni to’g’ri yozish.
2. Semantic teglardan to’g’ri foydalanish.
3. Meta teglardan to’g’ri foydalanish.
4. Social media’ga bog’lash.
5. Rasmlarga alt teglarini to’g’ri qo’yish.
6. Raqobatchi saytlardan namuna olish. (Yoki yaxshi saytlardan ham)

▼ Sayt xavfsizligi (*XSS, CSS injections, Code injections hujumlarini oldini olish*)

XSS(Cross site scripting) - Saytga begona bo’lgan script’larni kiritish orqali saytni buzishga urunish. Yechimlar: DOMPurify, HTMLSanitizier va hkz.

CSS Injections - Dasturga begona bo’lgan biror script’ni CSS orqali kiritish usuli. Ko’pincha background-image: url(”begonaUrl”); ni kiritish orqali bo’ladi. Hozirda yangi versiyali browser’lar allaqachon bu muommoni oldini olgan.

SQL Injections

Iframes - HTML’dagi Iframe tegini ishlatilgan holatda, saytimizga hujum qilish uchun yana bir imkon qoldirgan bo’lamiz. Bundan hacker’lar clickjacking hujumini qilishi ham mumkin.

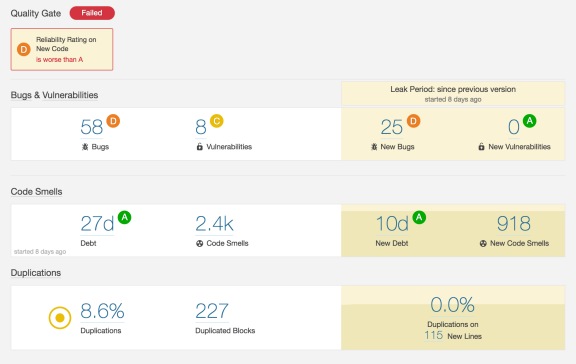
etc.

Maslahatlar

- Ishonchli third-party kutubxonalardan foydalaning.

- “Vulnerability”larni oldini oling. (npm audit va Sonarqube yoki shu kabi boshqa Vulnerability scanners’lardan foydalaning).

Example report:



Useful links

OWASP TOP 10: https://owasp.org/

Playground: xss.pwnfunction.com

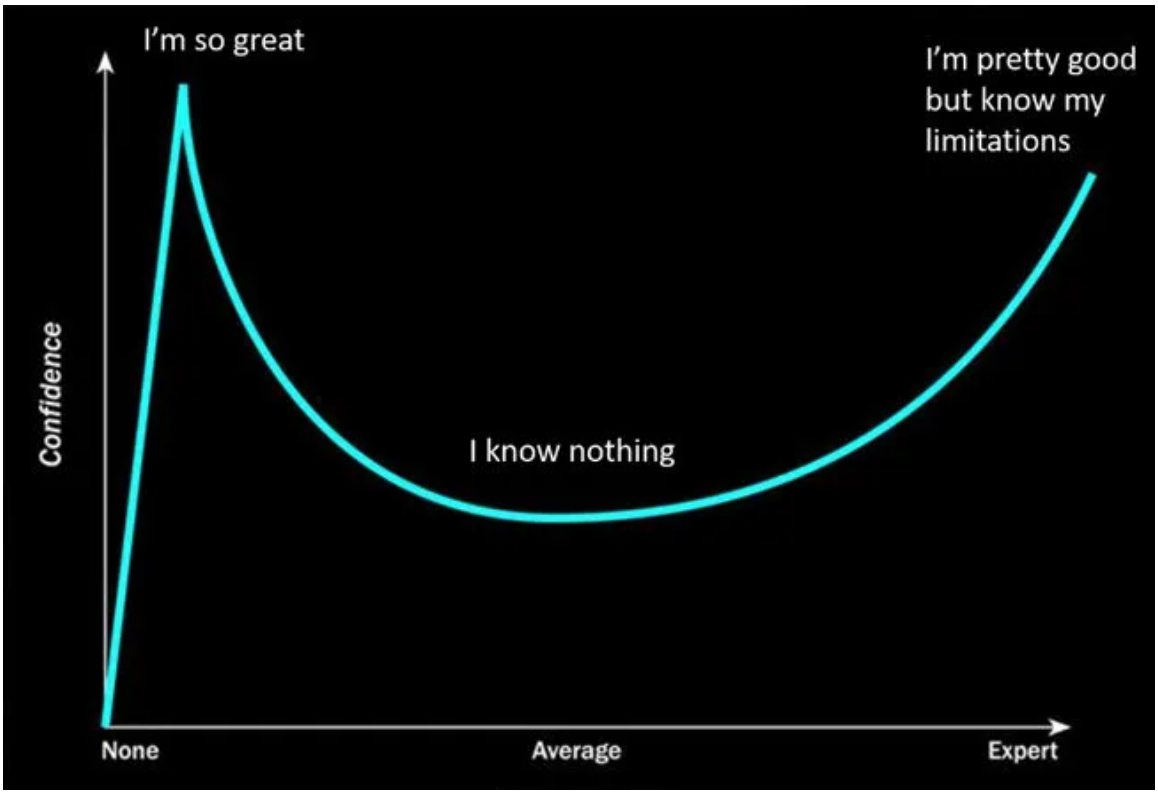
▼ Umumiy maslahatlar

▼ Yangi dars va ma’lumotlarni daftarga yozib boring. Erinmang.

▼ Dunning-kruger effekti.

Dunning-kruger effekti - inson o’zida bor bilimni ortiqcha baholashi va o’ziga haddan ortiq ishonib yuborishi va shundan keyingi jarayon.

Yangi narsa o’rganuvchilar uchun qo’llanma.



Video: https://www.youtube.com/watch?v=4FGnb2lgPBA

▼ Imposter sindromi.

Imposter sindromi - Inson o'ziga ishonmasligi, o'zini ayni damdagi ishiga loiq emas deb bilishi.

https://blog.hubspot.com/marketing/impostor-syndrome-tips

▼ Ikki kemada suza olmaysiz! (never sail on two boats)

▼ Google it!

1. Double quotes - "exact match".
2. two dots between range, 2001..2016
3. Searches inside specific site - site:smth.com
4. Excluding term with minus - -jquery
5. after some date, after:2012

6. before certain date, before:2012
7. filetype:pdf
8. related:angular
9. cache:url

▼ **Intervyularda savolingizni so’rashni unutmang.**

HR intervyuda kompaniya haqida, ishchilari, ish vaqtlari, oylik, umumiy loyihalar, ba’zi loyiha yoki topshiriqlar shaxsiy qadriyatlaringizga zid keladigan holatlarda kompaniya qanday javob qaytarishi kabi savollarni so’rang.

Texnik intervyu oxirida dasturchi sifatida o’zini kompaniya haqida fikrlarini so’rang. Intervyuda kutilgan natijani oldimi yoki yo’qmi so’rashingiz mumkin.

▼ **Islom dinini aralashtiravermang!**
▼ **Intizom eng asosiysi.**
▼ **Intervyu bu oddiy suhbat, vahima qilmang! (DON’T PANIC!)**
▼ **Oylikni gaplashish va oshirish sirlari.** 😎💸
▼ **Feedback olish va berish**
▼ **Estimation nima ?**

https://planningpokeronline.com/

▼ **Xotirada saqlash usullari.**
▼ **Ishda vazifalar olish va topshirish**

```
https://github.com/josh-developer/CI-CD-practice
```

▼ **Multitasking qilmang!**

▼ **Communication skillsni oshirish.**
▼ **Do not overwork!**