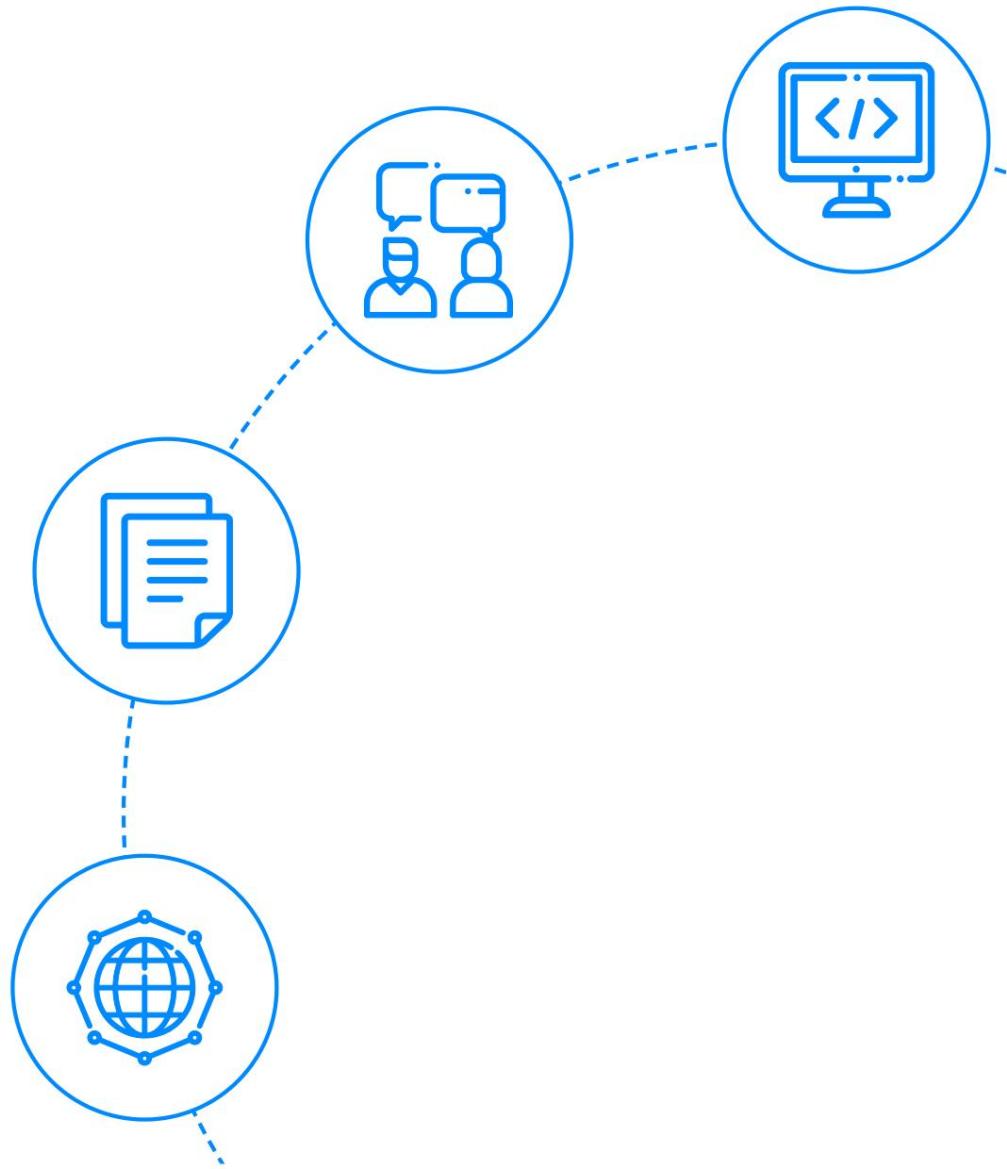




JavaScript intervyyu savollari



ning jonli versiyasini ko'rish uchun
sahifa, [shu yerni bosing](#).

Tarkib

Yangi talabalar uchun JavaScript Intervyu savollari

1. Javascriptda qanday ma'lumotlar turlari mavjud?
2. Javascriptda Hoistingni tushuntiring.
3. Nima uchun javascriptda "debugger" so'zini ishlatalamiz?
4. "" orasidagi farq == " va " === " operatorlar.
5. Javascriptdagi var va let kalit so'zlari orasidagi farq.
6. Javascriptda Implicit Type Coercionni tushuntiring.
7. Javascript statik yoki dinamik terilgan tilmى?
8. JavaScript-da NaN xossasi nima?
9. Qiymat bo'yicha o'tgan va mos yozuvlar bo'yicha o'tganlarni tushuntiring.
10. JavaScript-da darhol chaqiriladigan funksiya nima?
11. Javascriptda qat'iy rejim deganda nimani tushunasiz va JavaScript xususiyatlari qattiq rejimmi?
12. Javascriptda oliy tartibli funksiyalarni tushuntiring.
13. "Bu" kalit so'zini tushuntiring.
14. O'z-o'zini chaqiruvchi funksiyalar deganda nimani tushunasiz?
15. Call(), apply() va, bind() usullarini tushuntiring.
16. Javascriptdagi exec () va test () usullarining farqi nimada?
17. JavaScript-da karriing nima?
18. Tashqi JavaScript dan foydalanishning qanday afzalliklari bor?
19. Scope va Scope Chainni javascriptda tushuntiring.
20. JavaScript-da yopilishlarni tushuntiring.

JavaScript uchun intervyu savollari

Yangi kelganlar

(.....Davomi)

21. Javascriptning ba'zi afzalliklarini aytib o'ting.
22. Ob'ekt prototiplari nima?
23. Qayta qo'ng'iroqlar nima?
24. Javascriptda qanday xatolar turlari mavjud?
25. Memoizatsiya nima?
26. Dasturlash tilida rekursiya nima?
27. Javascriptda konstruktur funksiyasidan qanday foydalilanildi?
28. DOM nima?
29. Muayyan indeksdan belgini olish uchun qaysi usuldan foydalilanildi?
30. BOM deganda nimani tushunasiz?
31. Mijoz tomoni va server tomoni JavaScript-ning farqi nimada?

Tajribali uchun JavaScript Intervyu savollari

32. Ok funksiyalari nima?
33. Prototip dizayn namunasi deganda nimani tushunish kerak?
34. Var, let va const yordamida o'zgaruvchilarni e'lon qilish o'rtaqidagi farqlar.
35. Dam olish parametri va tarqalish operatori nima?
36. JavaScript-da ob'ektni necha xil usulda yasash mumkin?
37. Javascriptda va'dalar qanday qo'llaniladi?
38. Javascriptda qanday sinflar mavjud?

JavaScript uchun intervyu savollari

Tajribali

(.....Davomi)

- 39.** Generator funksiyalari nima?
- 40.** Javascriptda WeakSet ni tushuntiring.
- 41.** Nima uchun biz qayta qo'ng'iroqlardan foydalanamiz?
- 42.** WeakMap ni javascriptda tushuntiring.
- 43.** Obyekt tuzilmasini buzish nima?
- 44.** Prototip va klassik merosning farqi
- 45.** Vaqtinchalik o'lifik zona nima?
- 46.** JavaScript Design Patterns deganda nimani tushunasiz?
- 47.** JavaScript o'tish-by-reference yoki pass-by-value tilimi?
- 48.** Shunga erishish uchun Async/Await va Generatorlardan foydalanish o'rtaсидаги farq funksionallik.
- 49.** JavaScript-da qanday primitiv ma'lumotlar turlari mavjud?
- 50.** JavaScript-da kechiktirilgan skriptlarning roli qanday?
- 51.** Leksik qamrovni amalda qo'llash uchun nima qilish kerak?
- 52.** Quyidagi JavaScript kodining maqsadi nima?

JavaScript kodlash intervyu savollari

- 53.** Quyidagi kodlarning chiqishini taxmin qiling:
- 54.** Quyidagi kodning natijalarini taxmin qiling:
- 55.** Quyidagi kodning chiqishini taxmin qiling:
- 56.** Quyidagi kodning chiqishlarini taxmin qiling:

JavaScript-ni kodlash bo'yicha intervyu savollari (.....davom)

57. Quyidagi kodning natijalarini taxmin qiling:
58. Saralangan massivda ikkilik qidiruvni bajaradigan funksiyani yozing.
59. Yangilangan massivni r o'ngga aylantirish bilan qaytaruvchi funksiyani bajaring butun sonlar massivi a .
60. Yangi komponentlarni dinamik kiritish kodini yozing.
61. Berilgan kodni yozing Agar ikkita satr bir-birining anagrammasi bo'lsa, uni qaytaring rost.
62. Unli tovushlarni topish kodini yozing
63. JavaScript-da qanday qilib Ob'ektini Massiv [] ga aylantirasiz?
64. Quyidagi kodning chiqishi nima?

Keling, boshlaymiz

JavaScript, Brendan Eich tomonidan 1995 yilda yaratilgan, eng ko'p ishlatiladigan veb-ishlab chiqish tillaridan biri. Dastlab u dinamik veb-sahifalarni yaratish uchun mo'ljallangan edi. Skript - bu har qanday veb-sahifaning HTML-ga qo'shilishi mumkin bo'lgan JS dasturi. Sahifa yuklanganda, bu skriptlar avtomatik ravishda ishlaydi.

Dastlab dinamik veb-sahifalarni yaratish uchun mo'ljallangan til endi serverda va JavaScript Engine o'rnatilgan deyarli har qanday qurilmada ishga tushirilishi mumkin.

Aer HTML va CSS, JavaScript uchinchi yirik veb-texnologiya hisoblanadi. JavaScript - bu onlayn va mobil ilovalar, veb-serverlar, o'yinlar va boshqalarni yaratish uchun ishlatilishi mumkin bo'lgan skript tili. JavaScript - bu ob'ektga yo'naltirilgan dasturlash tili bo'lib, u veb-saytlar va ilovalar yaratish uchun ishlatiladi. U brauzerda foydalanish maqsadida yaratilgan. Bugungi kunda ham JavaScript-ning Node.js deb nomlanuvchi server tomonidagi versiyasi onlayn va mobil ilovalar, real vaqtda ilovalar, onlayn oqim ilovalari va video o'yinlar yaratish uchun ishlatilishi mumkin. **Javascript ramkalar**, o'rnatilgan kutubxonalar sifatida tanilgan, ish stoli va mobil dasturlarni yaratish uchun ishlatilishi mumkin. Ishlab chiquvchilar ushbu kod kutubxonalaridan foydalangan holda monoton dasturlash ishlariiga ko'p vaqt ni tejashlari mumkin, bu esa ularga ishlab chiqarishning ishlab chiqarish ishlariiga e'tibor qaratish imkonini beradi.



InterviewBit jamoasi sizga intervyu olishingiz va Javascript dasturchisi sifatida xohlagan ishingizga kirishda yordam berish uchun eng yaxshi **JavaScript intervyu savollari va javoblarining** to'liq to'plamini tuzdi .

Yangi talabalar uchun JavaScript Intervyu savollari

1. Javascriptda qanday ma'lumotlar turlari mavjud?

JavaScript o'zgaruvchisining turini bilish uchun **typeof** operatoridan foydalanishimiz mumkin.

1. Primitiv tiplar

String - u bir qator belgilarni ifodalaydi va tirnoq bilan yoziladi. Satr bitta yoki ikkita tirnoq yordamida ifodalanishi mumkin.

Misol:

```
var str = "Vivek Singx Bisht"; //qo'sh tirnoq yordamida var str2 =
'Jon Doe'; // bitta tirnoq yordamida
```

- **Raqam** - Bu raqamni ifodalaydi va o'nli yoki o'nli kasrlarsiz yozilishi mumkin.

Misol:

```
var x = 3; //o'nlik kasrsiz var y = 3,6; //
kasr bilan
```

- **BigInt** - Ushbu ma'lumotlar turi Raqamli ma'lumotlar turi chegarasidan yuqori bo'lgan raqamlarni saqlash uchun ishlataladi. U katta butun sonlarni saqlashi mumkin va butun son harfiga "n" qo'yishish orqali ifodalanadi.

Misol:

```
var bigInteger = 234567890123456789012345678901234567890;
```

- **Mantiqiy** - Bu mantiqiy ob'ektni ifodalaydi va faqat ikkita qiymatga ega bo'lishi mumkin: rost yoki noto'g'ri. Booleanlar odatda shartli sinov uchun ishlataladi.

Misol:

```
var a = 2; var
b = 3; var c =
2; (a == b) //
yolg'oni qaytaradi (a == c) // rostni
qaytaradi
```

- **Aniqlanmagan** - o'zgaruvchi e'lon qilingan, lekin tayinlanmagan bo'lsa, u undefined qiymatiga ega va uning turi ham aniqlanmagan.

Misol:

```
var x; // x ning qiymati aniqlanmagan var y =
aniqlanmagan; // o'zgaruvchining qiymatini aniqlanmagan deb ham belgilashimiz mumkin
```

- **Null** - u mavjud bo'lмаган yoki noto'g'ri qiymatni ifodalaydi.

Misol:

```
var z = null;
```

- **Symbol** - Bu javascriptning ES6 versiyasida kiritilgan yangi ma'lumotlar turi. U anonim va noyob qiymatni saqlash uchun ishlataladi.

Misol:

```
var symbol1 = Symbol('symbol');
```

- **ibtidoiy turlarning turlari :**

```
typeof "John Doe" // "string" turini qaytaradi typeof
3.14 // "raqam" turini qaytaradi true //
"mantiqiy" turini qaytaradi
23456789012345678901234567890n // Qaytaradi bigint typeof // turns
"undefined" turini qaytaradi . JavaScript-dagi xato turi)
typeof Symbol('symbol') // Symbolni qaytaradi
```

2. Primitiv bo'limgan turlar

- Primitiv ma'lumotlar turlari faqat bitta qiymatni saqlashi mumkin. Bir nechta va murakkab qiymatlarni saqlash uchun primitiv bo'limgan ma'lumotlar turlari qo'llaniladi.
- Ob'ekt - ma'lumotlar to'plamini saqlash uchun ishlataladi.
- Misol:

```
// Kalit-qiyomat juftliklarida ma'lumotlarni to'plash
```

```
var obj1 = { x:
  43, y:
  "Salom dunyo!", z:
  function(){ bu.x ;
    }
}
// Ma'lumotlarni tartiblangan ro'yxat sifatida to'plash

var array1 = [5, "Salom", rost, 4.1];
```

Eslatma - Shuni yodda tutish kerakki, ibtidoiy ma'lumotlar turi bo'lmagan har qanday ma'lumot turi JavaScript-da Ob'ekt turiga tegishli.

2. Javascriptda Hoistingni tushuntiring.

Yuk ko'tarish JavaScript-ning standart xatti-harakati bo'lib, unda barcha o'zgaruvchilar va funksiya deklaratsiyasi tepaga ko'chiriladi.

Declaration moves to top

a = 1;

alert(' a = ' + a);

var a;

Bu shuni anglatadiki, o'zgaruvchilar va funktsiyalar qayerda e'lon qilinganidan qat'i nazar, ular qamrovning yuqori qismiga ko'chiriladi. Qamrov ham mahalliy, ham global bo'lishi mumkin.

1-misol:

```
hoistedVariable = 3;
console.log(hoistedVariable); // o'zgaruvchidan keyin e'lon qilinganda ham 3 chiqadi var hoistedVariable;
```

2-misol:

```
hoistedFunction(); // Chiqishlar
```

"Salom dunyo!"

funksiya keyin e'lon qilinganda ham

```
funktsiya hoistedFunction(){
  console.log(" Salom dunyo! ");
}
```

3-misol:

```
// Yuk ko'tarish mahalliy miqyosda amalga oshiriladi, shuningdek doSomething ()x
= 33; console.log(x); var x;

}
```

doSomething(); // "x" mahalliy o'zgaruvchisi mahalliy doirada ko'tarilganligi sababli 33 chiqadi

Eslatma - O'zgaruvchilarni ishga tushirishlar ko'tarilmaydi, faqat o'zgaruvchan deklaratsiyalar ko'tariladi:

```
var x;
console.log(x); // "x" ning ishga tushirilishi ko'tarilmaganligi sababli "aniqlanmagan" chiqishlar x = 23;
```

Eslatma - Ko'tarilishning oldini olish uchun siz kodning tepasida "qat'iy foydalanish" dan foydalanib JavaScript-ni qattiq rejimda ishga tushirishingiz mumkin:

```
"qat'iy foydalaning";
x = 23; // Xato beradi, chunki 'x' e'lon qilinmagan var x;
```

3. Nima uchun javascriptda “debugger” so‘zini ishlatalamiz?

Kodni disk raskadrova qilish uchun brauzer uchun tuzatuvchi faollashtirilgan bo'lishi kerak. O'rnatilgan nosozliklarni tuzatuvchilarni yoqish va o'chirish mumkin, bu esa foydalanuvchidan nosozliklar haqida xabar berishni talab qiladi. The kodning qolgan qismi keyingisiga o'tishdan oldin bajarilishini to'xtatishi kerak disk raskadrova paytida chiziq.

4. "" orasidagi farq == " va " === " operatorlar.

Ikkalasi ham taqqoslash operatorlari. Ikkala operator, "o'rtasidagi farq shundaki "==" qiymatlarni solishtirish uchun ishlatiladi, "===" ikkala qiymatni solishtirish uchun ishlatiladi va "turlar.

Misol:

```
var x = 2;
var y = "2";
(x == y) // Haqiqatni qaytaradi, chunki x va y qiymati bir xil
(x === y) // noto'g'rini qaytaradi, chunki x turi "raqam" va y turi "string"
```

5. Javascriptdagi var va let kalit so'zlari orasidagi farq.

Ba'zi farqlar

1. Eng boshidan JavaScript dasturlashda 'var' kalit so'zi ishlatilgan **Holbuki, "kel" kalit so'zi** 2015 yilda qo'shilgan.
2. 'Var' kalit so'zi funksiya doirasiga ega. Funktsiyaning istalgan joyida, o'zgaruvchi var yordamida ko'rsatilgan ga kirish mumkin, lekin "let" bilan e'lon qilingan o'zgaruvchining doirasi "let" kalit so'zi u e'lon qilingan blok bilan cheklangan. a bilan boshlaylik Blok doirasi.
3. 'var' ko'tariladigan o'zgaruvchini e'lon qiladi, lekin 'let' o'zgaruvchini e'lon qiladi ko'tarilmoq.

6. Javascriptda Implicit Type Coercionni tushuntiring.

Javascriptdagi noaniq turdag'i majburlash qiymatni avtomatik ravishda birdan o'zgartirishdir ma'lumotlar turini boshqasiga o'tkazish. Bu ifoda operandlari bo'lganda sodir bo'ladi turli xil ma'lumotlar turlari.

- **String majburlash**

' operatoridan foydalanganda string majburlaş sodir bo'ladi. Raqam qo'shilganda satr, raqam turi har doim satr turiga aylantiriladi.

1-misol:

```
var x = 3;
var y = "3";
x + y // "33" ni qaytaradi
```

2-misol:

```
var x = 24;
var y = "Salom";
x + y // "24Salom" ni qaytaradi;
```

Eslatma - **+ operator ikkita raqamni qo'shish uchun ishlatsa, raqamni chiqaradi. The operator ikkita satr qo'shish uchun foydalanilganda, birlashtirilganni chiqaradi qator:**

```
var name = "Vivek";
var familiyasi =
    "Bisht";
ism + familiya // "Vivek Bisht" ni qaytaradi
```

Keling, qatorga raqam qo'shgan misollarni tushunaylik,

JavaScript x + y ifoda operandlari har xil turdag'i ekanligini ko'rganda (biri raqam turi, ikkinchisi esa satr turi), raqamni o'zgartiradi satr turiga kriting va keyin amalni bajaradi. Havo konvertatsiyasidan beri, ikkalasi ham o'zgaruvchilar qator tipida, birinchi **+** operator birlashtirilgan qatorni chiqaradi misolda "33" va ikkinchi misolda "24Salom".

Eslatma - "Farqni ishlatish paytida" operatoridan '-' operatori, lekin foydalanganda ham tur majburlash sodir bo'ladi, ya'ni satr raqamga aylantiriladi va keyin ayirish sodir bo'ladi.

```
var x = 3;
Var y = "3";
x - y      //y o'zgaruvchisi (satr turi) raqam turiga aylantirilgani uchun 0 ni qaytaradi
```

- **Boolean majburlash**

Mantiqiy majburlash mantiqiy operatorlardan, uchlik operatorlardan foydalanganiga sodir bo'ladi, agar bayonotlar va loop tekshiruvlari. If iboralarida mantiqiy majburlashni tushunish va operatorlar, biz haqiqat va yolg'on qadriyatlarni tushunishimiz kerak.

Haqiqiy qadriyatlar - bu haqiqatga aylantiriladigan (majburiy) bo'lganlar . Soxta qiymatlar yolg'onga aylantiriladiganlar .

False, 0, On, -0, "", null, undefined va NaN dan tashqari barcha qiymatlar rost qiymatlardir.

Agar bayonotlar:

Misol:

```
var x = 0;
var y = 23;

if(x) { console.log(x) } // Ushbu blok ichidagi kod o qiymatidan beri ishlamaydi.

if(y) { console.log(y) }      // Ushbu blok ichidagi kod y qiymatidan boshlab ishlaydi
```

- **Mantiqiy operatorlar:**

Javascriptdagи mantiqiy operatorlar boshqa dasturlash tillaridagi operatorlardan farqli o'laroq, **rost yoki yolg'oni qaytarmaydi. Ular har doim operandlardan birini qaytaradilar.**

OR (||) operatori - Agar birinchi qiymat rost bo'lса, birinchi qiymat qaytariladi.

Aks holda, har doim ikkinchi qiymat qaytariladi.

AND (&&) operatori - Agar ikkala qiymat ham to'g'ri bo'lса, har doim ikkinchi qiymat qaytariladi. Agar birinchi qiymat noto'g'ri bo'lса, birinchi qiymat qaytariladi yoki ikkinchi qiymat noto'g'ri bo'lса, ikkinchi qiymat qaytariladi.

Misol:

```
var x = 220; var
y = "Salom"; var z =
aniqlanmagan;

x | y // 220 ni qaytaradi, chunki birinchi qiymat haqiqatdir
x | z // 220 ni qaytaradi, chunki birinchi qiymat haqiqatdir
x && y // "Salom"ni qaytaradi, chunki ikkala qiymat ham haqiqatdir
y && z // noaniq qaytariladi, chunki ikkinchi qiymat noto'g'ri

if( x && y )
{ console.log("Kod ishlaydi" ); // Bu blok ishlaydi, chunki x && y "Salom" (Haqiqat) ni qaytaradi }

if( x || z )
{ console.log("Kod ishlaydi"); // Bu blok ishlaydi, chunki x || y 220 (Haqiqat) ni qaytaradi }
```

- **Tenglikka majburlash**

Foydalanishda tenglik majburlash sodir bo'ladi `' == '` operator. Avval aytib o'tganimizdek

`' == '` operator turlarni emas, balki qiymatlarni taqqoslaydi.

Yuqoridagi bayonot `==` operatorini tushuntirishning oddiy usuli bo'lsa-da, bu to'liq emas rost

Haqiqat shundaki, `'=='` operatoridan foydalanganda majburlash sodir bo'ladi.

`'=='` operatori ikkala operandni bir xil turga aylantiradi va keyin solishtiradi ular.

Misol:

```
var a = 12;
var b = "12";
a == b // rostni qaytaradi, chunki "a" va "b" bir xil turga aylantiriladi va keyin
```

`'==='` operatoridan foydalanganda majburlash amalga oshirilmaydi. Ikkala operand ham emas `'==='` operatori holatida bir xil turga aylantiriladi.

Misol:

```
var a = 226;
var b = "226";
a === b // falseni qaytaradi, chunki majburlash sodir bo'lmaydi va operandlar
```

7. Javascript statik yoki dinamik terilganmi? til?

JavaScript dinamik ravishda terilgan tildir. Dinamik tarzda terilgan tilda o'zgaruvchining turi statik tarzda kiritilgandan farqli ravishda **ish vaqtida** tekshiriladi **kompilyatsiya** vaqtida o'zgaruvchining turi tekshiriladigan til .

Static Typing

```
string name;
name = "John";
name = 34;
```

Variables have types

Values have types

Variables cannot change type

Dynamic Typing

```
var name;
name = "John";
name = 34;
```

Variables have no types

Values have types

Variables change type dramatically



Javascript erkin (dinamik) terilgan til bo'lgani uchun JSdagi o'zgaruvchilar hech qanday tur bilan bog'lanmaydi. O'zgaruvchi har qanday turdag'i ma'lumotlarning qiymatini ushlab turishi mumkin.

Misol uchun, raqam turi tayinlangan o'zgaruvchini satr turiga aylantirish mumkin:

```
var a = 23; var
a = "Salom dunyo!";
```

8. JavaScript-da NaN xossasi nima?

NaN xususiyati "**Raqam emas**" qiymatini ifodalaydi. Bu qonuniy raqam bo'limgan qiymatni bildiradi.

typeof NaN raqamni qaytaradi .

Qiymat NaN ekanligini tekshirish uchun **isNaN()** funksiyasidan foydalanamiz,

Eslatma- **isNaN()** funksiyasi berilgan qiymatni Raqam turiga aylantiradi va keyin NaN ga tenglashadi.

```
isNaN("Salom") // true qaytaradi
isNaN(345) // false isNaN('1')
qaytaradi // Noto'g'ri qaytaradi, chunki '1' Raqam turiga aylantiriladi, natijada 0 isNaN(true) // noto'g'ri qaytaradi ,
chunki true soni Raqam turiga aylantirilganda natija 1 bo'ladi ( numb isNaN(noto'g'ri) // false isNaN(aniqlanmagan)
qaytaradi // true qiymatini qaytaradi
```

9. Qiymat bo'yicha o'tgan va mos yozuvlar bo'yicha o'tganlarni tushuntiring.

JavaScript-da ibtidoiy ma'lumotlar turlari qiymat bo'yicha, noaniq ma'lumotlar turlari esa mos yozuvlar bo'yicha uzatiladi.

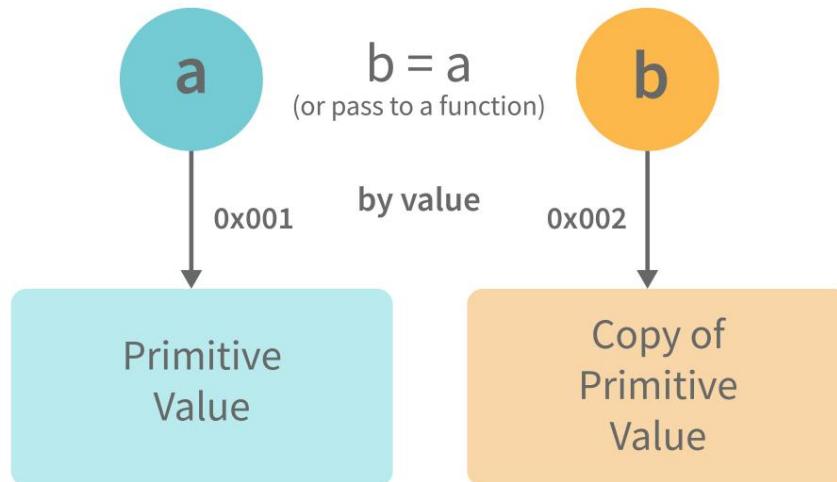
Qiymat bo'yicha o'tgan va mos yozuvlar orqali o'tgan tushunish uchun biz o'zgaruvchini yaratganimizda va unga qiymat berganimizda nima sodir bo'lishini tushunishimiz kerak,

```
var x = 2;
```

Yuqoridagi misolda biz x o'zgaruvchisini yaratdik va unga "2" qiymatini berdik. Orqa fonda "=" (tayinlash operatori) xotirada bir oz joy ajratadi, "2" qiymatini saqlaydi va ajratilgan xotira maydonining o'rnnini qaytaradi. Shuning uchun, yuqoridagi koddagi x o'zgaruvchisi to'g'ridan-to'g'ri 2 qiymatini ko'rsatish o'rniiga xotira maydonining joylashgan joyiga ishora qildi.

Assign operatori ibtidoiy va ibtidoiy bo'Imagan ma'lumotlar turlari bilan ishlashda o'zini boshqacha tutadi,

Primitiv turlar bilan ishlaydigan operatorni tayinlang:



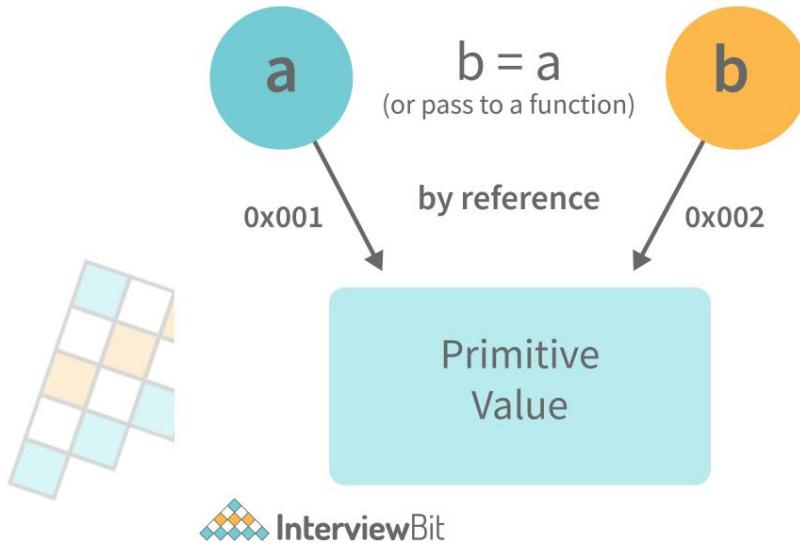
```
var y = 234; var z
= y;
```

Yuqorida misolda tayinlash operatori y ga tayinlangan qiymat ibtidoiy tip ekanligini biladi (bu holda raqam turi), shuning uchun ikkinchi qator kodi bajarilganda, y qiymati z ga tayinlanganda, tayinlash operatori y qiymati (234) va xotirada yangi joy ajratadi va manzilni qaytaradi. Shuning uchun, z o'zgaruvchisi y o'zgaruvchining joylashgan joyini emas, balki xotiradagi yangi joyni ko'rsatmoqda.

```
var y = #8454; // y qiymati 234 manziliga ishora qiladi
var z = y;
var z = #5411; // z 234 qiymatining mutlaqo yangi manziliga ishora qiladi
// y qiymatini o'zgartirish y = 23;
console.log(z); // 234 ni qaytaradi, chunki z xotiradagi yangi manzilga ishora qiladi, shuning uchun o'zgaradi
```

Yuqoridagi misoldan ko'rishimiz mumkinki, ibtidoiy ma'lumotlar turlari boshqa o'zgaruvchiga o'tkazilganda qiymat bo'yicha uzatiladi. Xuddi shu manzilni boshqa o'zgaruvchiga belgilash o'rniغا, qiymat uzatiladi va yangi xotira maydoni yaratiladi.

Primitiv bo'limgan turlar bilan ishlaydigan operatorni tayinlang:



```
var obj = { ism: "Vivek", familiya: "Bisht" }; var obj2 = obj;
```

Yuqoridagi misolda assign operatori obj o'zgaruvchisining joylashuvini to'g'ridan-to'g'ri obj2 o'zgaruvchisiga uzatadi. Boshqacha qilib aytganda, obj o'zgaruvchisiga havola obj2 o'zgaruvchisiga uzatiladi.

```
var obj = #8711; // obj manziliga ishora qiluvchi { name: "Vivek", familiya: "Bisht" } var obj2 = obj;
```

```
var obj2 = #8711; // obj2 bir xil manzilga ishora qiladi
```

```
// obj1 qiyimatini o'zgartirish
```

```
obj1.name = "Akki";  
console.log(obj2);
```

// {name:"Akki", familiya:"Bisht"} qaytaradi, chunki ikkala o'zgaruvchi ham o'zgaruvchiga ishora qiladi.

Yuqoridagi misoldan ko'rishimiz mumkinki, primitiv bo'limgan ma'lumotlar turlarini o'tkazishda tayinlash operatori to'g'ridan-to'g'ri manzilni (ma'lumotnoma) uzatadi.

Shuning uchun primitiv bo'limgan ma'lumotlar turlari har doim **mos yozuvlar orqali** uzatiladi.

10. JavaScript-da darhol chaqiriladigan funksiya nima?

Darhol chaqiriladigan funksiya (IIFE deb nomlanadi va IIFY deb talaffuz qilinadi) bu aniqlangandan so'ng darhol ishlaydigan funksiya.

IIFE sintaksisi:

```
(funktsiya(){  
    // Biror narsa qiling; }  
());
```

IIFE-ni tushunish uchun biz IIFE yaratishda qo'shiladigan ikkita qavs to'plamini tushunishimiz kerak:

Birinchi qavslar to'plami:

```
(funktsiya (){  
    //Biror narsa qilish;  
})
```

Javascript kodini bajarayotganda, kompilyator "funktsiya" so'zini ko'rganda, biz kodda funktsiyani e'lon qilyapmiz deb taxmin qiladi. Shuning uchun, birinchi qavslar to'plamidan foydalanmasak, kompilyator biz funktsiyani e'lon qilyapmiz deb o'ylaganligi sababli xatoga yo'l qo'yadi va funktsiyani e'lon qilish sintaksisi bo'yicha funksiya har doim nomga ega bo'lishi kerak.

```
funktsiya() {
    //Biror narsa qilish; }

// Kompilyator xato beradi, chunki kodda funktsiyani e'lon qilish sintaksisi noto'g'ri
```

Ushbu xatoni olib tashlash uchun biz kompilyatorga funksiya funksiya deklaratsiyasi emas, balki funksiya ifodasi ekanligini bildiruvchi birinchi qavslar to'plamini qo'shamiz.

Ikkinci qavs to'plami:

```
(funktsiya () {
    //Biror narsa qilish; })();
```

IIFE ta'rifidan bilamizki, bizning kodimiz aniqlangan zahoti ishga tushishi kerak. Funktsiya faqat chaqirilganda ishlaydi. Agar funktsiyani chaqirmasak, funktsiya deklaratsiyasi qaytariladi:

```
(funktsiya () {
    // Biror narsa qiling; })()

// Funktsiya deklaratsiyasini qaytaradi
```

Shuning uchun funktsiyani chaqirish uchun biz ikkinchi qavs to'plamidan foydalanamiz.

11. Javascriptdagi qat'iy rejim va javascriptning qat'iy rejimi deganda nimani tushunasiz?

ECMAScript 5 da JavaScript Strict Mode deb nomlanuvchi yangi xususiyat “qattiq” operatsion muhitda kod yoki funksiya yozish imkonini beradi. Ko'pgina hollarda, bu til xatolarga yo'l qo'yishda "ayniqsa og'ir emas". Biroq, "Qat'iy rejimda" barcha xatolar, jumladan, ovozsiz xatolar ham tashlanadi. Natijada, disk raskadrovska ancha soddalashadi. Shunday qilib, dasturchining xato qilish ehtimoli kamayadi.

Javascriptdagi qat'iy rejimning xususiyatlari

1. Ikki nusxadagi argumentlar ishlab chiquvchilar tomonidan ruxsat etilmaydi.
2. Qattiq rejimda siz JavaScript kalit so'zidan parametr yoki funksiya nomi sifatida foydalana olmaysiz.
3. "use strict" kalit so'zi skript boshida qat'iy rejimni aniqlash uchun ishlatiladi.
Qattiq rejim barcha brauzerlar tomonidan qo'llab-quvvatlanadi.
4. Muhandislarga "Qat'iy rejim"da global o'zgaruvchilar yaratishga ruxsat berilmaydi.

12. Javascriptda oliy tartibli funksiyalarni tushuntiring.

Boshqa funktsiyalarda yoki ularni argument sifatida qabul qilish yoki ularni qaytarish orqali ishlaydigan funktsiyalar yuqori tartibli funktsiyalar deb ataladi.

Yuqori darajadagi funktsiyalar javascriptda **birinchi darajali fuqarolar** bo'lgan funktsiyalarning natijasidir .

Yuqori darajadagi funktsiyalarga misollar:

```
function moreOrder(fn) { fn(); }

highOrder(function() { console.log("Salom dunyo") });
```

```

function laterOrder2() { return
  function() { return "Biror
    narsa qiling";
  }
}

} var x = laterOrder2(); x() //
"Biror narsa qilish" ni qaytaradi

```

13. “Bu” kalit so‘zini tushuntiring.

“This” kalit so‘zi funksiya xossasi bo‘lgan obyektga ishora qiladi.

“This” kalit so‘zining qiymati har doim funksiyani chaqirayotgan obyektga bog‘liq bo‘ladi.

\

Adashib qoldingizmi? Keling, yuqoridaq gaplarni misollar bilan tushunamiz:

```

function doSomething()
{ console.log(bu); }

doSomething();

```

Sizningcha, yuqoridaq kodning chiqishi qanday bo‘ladi?

Eslatma - Funktsiyani chaqirayotgan qatorga e’tibor bering.

Ta’rifni yana tekshiring:

“This” kalit so‘zi funksiya xossasi bo‘lgan obyektga ishora qiladi.

Yuqoridaq kodda funksiya qaysi obyektning xususiyati hisoblanadi?

Funktsiya global kontekstda chaqirilganligi sababli, **funktsiya global ob'ektning xossasidir.**

Shuning uchun yuqoridaq kodning chiqishi **global ob'ekt bo'ladi**. Yuqoridaq kodni brauzer ichida ishlatganimiz uchun global ob'ekt **oyna ob'ekti**dir.

2-misol:

```
var obj =
  { name: "vivek",
    getName: function()
    { console.log(this.name); }

  }
obj.getName();
```

Yuqoridagi kodda, chaqirish vaqtida getName funksiyasi **obj** ob'ektining xossasidir , shuning uchun **bu** kalit so'z **obj** ob'ektiga tegishli bo'ladi va shuning uchun chiqish "vivek" bo'ladi.

3-misol:

```
var obj =
  { name: "vivek",
    getName: function()
    { console.log(this.name); }

  }
var getName = obj.getName;

var obj2 = {name:"akshay", getName };
obj2.getName();
```

Bu erda chiqishni taxmin qila olasizmi?

Chiqish "akshay" bo'ladi.

getName funksiyasi obj ob'ektida e'lon qilingan bo'lsa-da , chaqirish vaqtida getName() **obj2 ning xossasidir**, shuning uchun "this" kalit so'zi **obj2 ga tegishli bo'ladi**.

"**Bu**" kalit so'zini tushunishning ahmoqona usuli shundaki, har doim funksiya chaqirilganda, ob'ektni nuqtadan oldin tekshiring. **Buning** qiymati . kalit so'z har doim nuqtadan oldin ob'ekt bo'ladi .

Agar 1-misolda nuqtaga o'xshash ob'ekt bo'lmasa, bu kalit so'zning qiymati global ob'ekt bo'ladi.

4-misol:

```
var obj1 =
  { manzil : "Mumbay, Hindiston",
    getAddress: function()
    { console.log(this.address); }

  }

var getAddress = obj1.getAddress; var obj2
= {name:"akshay"}; obj2.getAddress();
```

Chiqishni taxmin qila olasizmi?

Chiqish xato bo'ladi.

Yuqoridagi kodda bu kalit so'z **obj2 ob'ektiga tegishli bo'lسا-da**, obj2 "manzil" xususiyatiga ega emas, shuning uchun getAddress funksiyasi xatoga yo'l qo'yadi.

14. O'z-o'zini chaqiruvchi funksiyalar deganda nimani tushunasiz?

So'ralmagan holda, o'z-o'zini chaqiruvchi ifoda avtomatik ravishda chaqiriladi (boshlanadi).

Agar funksiya ifodasidan keyin () bo'lisa, u avtomatik ravishda bajariladi. Funktsiya deklaratsiyasini o'zi chaqirib bo'lmaydi.

Odatda, biz funksiyanı e'lon qilamiz va uni chaqiramiz, biroq, funksiya tasvirlanganda avtomatik ravishda ishga tushirish uchun anonim funksiyalardan foydalanish mumkin va qayta chaqirilmaydi. Va bu turdagı funksiyalar uchun hech qanday nom yo'q.

15. Call(), apply() va, bind() usullarini tushuntiring.

1. qo'ng'iroq ():

- Bu javascriptda oldindan belgilangan usul.
- Bu usul ega ob'ektini ko'rsatish orqali usulni (funksiyanı) chaqiradi.
- 1-misol:

```
function sayHello(){ "Salom" +
  this.nameni qaytarish ;
}
```

```
var obj = {ism: "Sandy"};
```

```
sayHello.call(obj);
```

// "Salom Sandy"ni qaytaradi

- call() usuli ob'ektga boshqa ob'ektning usulidan (funktsiyasidan) foydalanishga imkon beradi.
- 2-misol:

```
var person = {age: 23,
  getAge:
    function(){ return this.age;
  }
}
var person2 = {yosh: 54};
person.getAge.call(person2); // 54-ni qaytaradi
```

- call() argumentlarni qabul qiladi:

```
funksiya saySomething(xabar){ + xabar;
  this.name + ni qaytaring "bu"
}
var person4 = {ism: "Jon"};
saySomething.call(shaxs4, "ajoyib"); // "Jon ajoyib" ni qaytaradi
```

amal qilish()

Qo'llash usuli call() usuliga o'xshaydi. Faqatgina farq shundaki,

call() usuli argumentlarni alohida oladi, application() usuli esa argumentlarni massiv sifatida oladi.

```
funktsiya saySomething(xabar){
    this.name + " is " + xabar ;
}
var person4 = {ism: "Jon"};
saySomething.apply(shaxs4, ["ajoyib"]);
```

2. bog'lash():

- Ushbu usul yangi funktsiyani qaytaradi, bu erda "**bu**" kalit so'zining qiymati bo'ladi parametr sifatida taqdim etilgan egasi ob'ektiga bog'langan.
- Argumentlar bilan misol:

```
var bikeDetails = {
    display tafsilotlari: funktsiya (registrationNumber,brandName){
        this.name+ ni qaytaring "      , " + "velosiped tafsilotlari: "+ ro'yxatga olish raqami + " , " + brend nomi;
    }
}

var person1 = {ism: "Vivek"};

var detailsOfPerson1 = bikeDetails.displayDetails.bind(person1, "TS0122", "O'q");

// displayDetails funksiyasini person1 ob'ektiga bog'laydi

detailsOfPerson1();
// Vivekni qaytaradi, velosiped tafsilotlari: TS0452, Thunderbird
```

16. Exec () va test () usullarining farqi nimada javascript?

- test ()** va **exec ()** javascriptda ishlataladigan RegExp ifoda usullaridir.
- Muayyan naqsh uchun satrni qidirish uchun **exec () dan** foydalanamiz va agar u topsa, u naqshni to'g'ridan-to'g'ri qaytarish; aks holda, u "bo'sh" natijani qaytaradi.
- Muayyan naqsh uchun satrni topish uchun **test () dan** foydalanamiz . ni qaytaradi Berilgan matnni topishda mantiqiy qiymat "true" bo'lsa, u "noto'g'ri"ni qaytaradi.

17. JavaScript-da karriing nima?

Currying - bu n argumentlar funksiyasini n ga aylantirishning ilg'or usuli bir yoki bir nechta argumentlarning funktsiyalari.

Kurrlangan funktsiyaga misol:

```
funktsiya qo'shish (a) {
    qaytish funktsiyasi(b)
    { a + b qaytarish ;
    }
}

qo'shish (3) (4)
```

Misol uchun, agar bizda **f(a,b)** funksiyasi bo'lsa , u holda aer currying funktsiyasi **f(a)(b)** ga o'zgaradi .

Karriing texnikasidan foydalanib, biz funktsiyaning funksionalligini o'zgartirmaymiz, shunchaki uni chaqirish usulini o'zgartiramiz.

Keling, karrini amalda ko'rib chiqaylik:

```
funktsiyani ko'paytirish(a,b)
{ a*bni qaytarish ;
}

funktsiya currying(fn)
{ funktsiyani qaytarish (a){
    qaytish funktsiyasi(b)
    { fn(a,b) qaytarish ;
    }
}
}

var curriedMultiply = currying(ko'paytirish);

ko'paytirish (4, 3); // 12-ni qaytaradi

curriedMultiply(4)(3); // Shuningdek, 12 ni qaytaradi
```

Yuqoridagi koddan ko'rilib turibdiki, biz **multiply(a,b)** funksiyasini bir vaqtning o'zida bitta parametrni qabul qiluvchi **curriedMultiply** funksiyasiga aylantirdik.

18. Tashqi JavaScript dan foydalanishning qanday afzalliklari bor?

Tashqi JavaScript bu kengaytmali alohida faylda yozilgan JavaScript kodi (skript).

Tashqi javascriptning ba'zi afzalliklari

1. Bu veb-dizaynerlar va ishlab chiquvchilarga HTML va javascriptda hamkorlik qilish imkonini beradi fayllar.
2. Biz kodni qayta ishlatsizimiz mumkin.
3. Kodning o'qilishi tashqi javascriptda oddiy.

19. Scope va Scope Chainni javascriptda tushuntiring.

JSdagi qamrov o'zgaruvchilar va funktsiyalarning kodning turli qismlarida foydalanish imkoniyatini aniqlaydi.

Umuman olganda, qamrov bizga kodning ma'lum bir qismida qanday o'zgaruvchilar va funktsiyalarga kirishimiz yoki kira olmasligimiz haqida ma'lumot beradi.

JSda uch xil doiralar mavjud:

- Global qamrov
- Mahalliy yoki funksiya doirasi
- Blok doirasi

Global qamrov: Global nomlar maydonida e'lon qilingan o'zgaruvchilar yoki funktsiyalar global qamrovga ega, ya'ni global miqyosga ega bo'lgan barcha o'zgaruvchilar va funktsiyalarga kodning istalgan joyidan kirish mumkin.

```
var globalVariable = "Salom dunyo";

sendMessage() funksiyasi
{ globalVariableni qaytaradi ; // globalVariable-ga kirish mumkin, chunki u global oraliqda yozilgan

} sendMessage2() funksiyasi
{ sendMessage (); // SendMessage funksiyasiga kirish mumkin, chunki u global tilda yozilgan

} sendMessage2(); // "Salom dunyo"ni qaytaradi
```

Funktsiya doirasi: Funktsiya ichida e'lon qilingan har qanday o'zgaruvchilar yoki funktsiyalar mahalliy/funktsiya doirasiga ega, ya'ni funktsiya ichida e'lon qilingan barcha o'zgaruvchilar va funktsiyalarga funktsiyadan tashqarida emas, balki uning ichidan kirish mumkin.

```
function awesomeFunction(){ var a
    = 2;

    var multiplyBy2 = function()
        { console.log(a*2); // "a" o'zgaruvchisiga kirish mumkin, chunki a va multiplyBy2 ikkalasi ham yoziladi }

} console.log(a); // Malumot xatosini chiqaradi, chunki a mahalliy miqyosda yozilgan va mumkin emas

multiplyBy2(); // multiplyBy2 mahalliy miqyosda yozilgani uchun mos yozuvlari xatosini chiqaradi
```

Blok doirasi: Blok doirasi let va const yordamida e'lon qilingan o'zgaruvchilar bilan bog'liq. var bilan e'lon qilingan o'zgaruvchilar blok doirasiga ega emas. Blok doirasi bizga {} blokida e'lon qilingan har qanday o'zgaruvchiga faqat shu blok ichida kirish mumkinligini va undan tashqarida kirish mumkin emasligini aytadi.

```
{
    x = 45 bo'sin ;
}

console.log(x); // Malumot xatosi beradi, chunki x ga blokdan tashqarida kirish mumkin emas

for(let i=0; i<2; i++){ // biror narsa
    qilish }

console.log(i); // Malumot xatosi beradi, chunki i ga for dan tashqari kirish mumkin emas
```

Scope Chain: JavaScript mexanizmi o'zgaruvchilarni topish uchun Scope-dan ham foydalanadi. Keling, buni misol yordamida tushunaylik:

```

var y = 24;

funktsiya favFunction(){ var x
= 667; var
anotherFavFunction = function(){}
console.log(x); // anotherFavFunction ichida x topilmadi, shuning uchun o'zgaruvchini qidiradi }

var yetAnotherFavFunction = function(){}
console.log(y); // HaliAnotherFavFunction ichida y topilmadi, shuning uchun varia qidiradi }

anotherFavFunction();
yetAnotherFavFunction(); }

favFunction();

```

Yuqoridagi kodda ko'rib turganingizdek, agar JavaScript mexanizmi o'zgaruvchini mahalliy miqyosda topa olmasa, u o'zgaruvchini tashqi doirada tekshirishga harakat qiladi. Agar o'zgaruvchi tashqi miqyosda mavjud bo'lmasa, u o'zgaruvchini global miqyosda topishga harakat qiladi.

Agar o'zgaruvchi global maydonda ham topilmasa, mos yozuvlar xatosi tashlanadi.

20. JavaScript-da yopilishlarni tushuntiring.

Yopish - bu funksianing tashqi doirasida e'lon qilingan o'zgaruvchilar va funktsiyalarni eslab qolish qobiliyati.

```

var Person = function(pName){ var name
= pName;

this.getName = function(){ nomini
qaytarish ;
}
}

var person = new Person("Neelesh");
console.log(person.getName());

```

Keling, misol orqali yopilishni tushunaylik:

```

function randomFunc()
  { var obj1 = {ism:"Vivian", yosh:45};

  return function()
    { console.log(obj1.name + " bu "+ "ajoyib"); // Ishlaganda ham obj1 ga kirish huquqiga ega

    }
  }

var initialiseClosure = randomFunc(); // Funktsiyani qaytaradi

initialiseClosure();

```

Keling, yuqoridagi kodni tushunaylik,

RandomFunc() funktsiyasi bajariladi va biz uni o'zgaruvchiga tayinlaganimizda funktsiyani qaytaradi:

```
var initialiseClosure = randomFunc();
```

Biz initialiseClosure ni chaqiranimizda qaytarilgan funksiya bajariladi:

```
initialiseClosure();
```

Yuqoridagi kod qatori "Vivian ajoyib" degan xulosaga keladi va bu yopilganligi sababli mumkin.

```
console.log(obj1.name + " bu "+ "ajoyib");
```

RandomFunc() funktsiyasi ishga tushganda, qaytaruvchi funksiya uning ichidagi obj1 o'zgaruvchisidan foydalanayotganga o'xshaydi:

Shuning uchun randomFunc(), obj1 aer bajarilishining qiymatini yo'q qilish o'rniغا, **keyingi ma'lumot uchun qiymatni xotiraga saqlaydi**. Aynan shuning uchun qaytib keladigan funksiya tashqi miqyosda e'lon qilingan o'zgaruvchidan funksiya allaqachon bajarilgan bo'lsa ham foydalanishi mumkin.

Funktsiyaning o'zgaruvchini bajarilgandan keyin ham qo'shimcha ma'lumot olish uchun saqlash qobiliyati Yopish deb ataladi.

21. Javascriptning ba'zi afzalliklarini aytib o'ting.

Javascriptning afzalliklari juda ko'p. Ulardan ba'zilari

1. Javascript mijoz tomonidan ham, server tomonida ham bajariladi. Siz o'rganishingiz va foydalanishingiz mumkin bo'lgan turli xil Frontend Frameworklar mavjud. Biroq, agar siz JavaScript-ni backendda ishlatmoqchi bo'lsangiz, NodeJS-ni o'rganishingiz kerak bo'ladi. Hozirda bu backendda ishlatalishi mumkin bo'lgan yagona JavaScript ramkasi.
2. Javascript o'rganish uchun oddiy tildir.
3. Javascript tufayli veb-sahifalar endi ko'proq funksionallikka ega.
4. Yakuniy foydalanuvchi uchun Javascript juda tez.

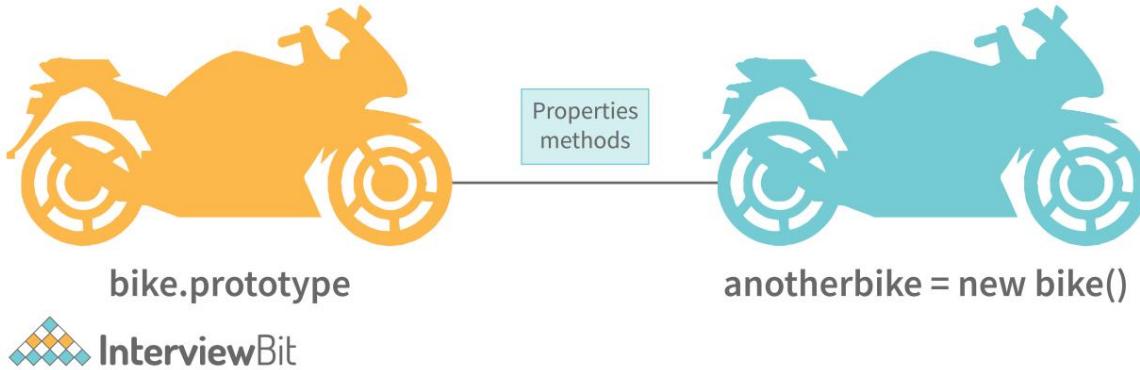
22. Ob'ekt prototiplari nima?

Barcha javascript obyektlari prototipdan xossalarni meros qilib oladi. Masalan,

- Sana obyektlari Sana prototipidagi xususiyatlarni meros qilib oladi
- Math obyektlari Math prototipidan xususiyatlarni meros qilib oladi
- Massiv obyektlari Array prototipidan xususiyatlarni meros qilib oladi.
- Zanjirning tepasida **Object.prototype joylashgan**. Har bir prototip Object.prototype dan xossa va usullarni meros qilib oladi.
- **Prototip - bu ob'ektning rejasi. Prototip**, hozirgi ob'ektda xususiyatlar va usullar mavjud bo'limasa ham, ob'ektda xususiyatlar va usullardan foydalanishga imkon beradi.

Keling, prototiplarni usullar va xususiyatlardan foydalanishga yordam beraylik:

OBJECT prototypes


 InterviewBit

```
var arr = [];
arr.push(2);

console.log(arr); // Chiqishlar [2]
```

Yuqoridagi kodda, ko'rib turganingizdek, biz "arr" massivida push deb nomlangan xususiyat yoki usulni aniqlamadik, ammo JavaScript mexanizmi xato qilmaydi.

Sababi prototiplardan foydalanish. Yuqorida aytib o'tganimizdek, Array ob'ektlari Array prototipidan xususiyatlarni meros qilib oladi.

Javascript mexanizmi joriy massiv ob'ektida surish usuli mavjud emasligini ko'radi va shuning uchun Array prototipi ichidan push usulini qidiradi va u usulni topadi.

Joriy ob'ektda xususiyat yoki usul topilmasa, JavaScript mexanizmi har doim o'z prototipini ko'rishga harakat qiladi va agar u hali ham mavjud bo'lmasa, u prototip prototipining ichiga qaraydi va hokazo.

23. Qayta qo'ng'iroqlar nima?

Qayta qo'ng'iroq - bu boshqa funktsiya bajarilgandan keyin bajariladigan funktsiya. JavaScript-da funksiyalar birinchi darajali fuqarolar sifatida ko'rib chiqiladi, ular boshqa funktsiyaning argumenti sifatida ishlatalishi mumkin, boshqa funktsiya tomonidan qaytarilishi mumkin va ob'ektning mulki sifatida ishlatalishi mumkin.

Boshqa funktsiyaga argument sifatida ishlataladigan funksiyalar qayta qo'ng'iroq qilish funksiyalari deb ataladi. Misol:

```
funktsiya divideByHalf(sum)
{ console.log(Math.floor(sum / 2)); }
```

```
funktsiya multiplyBy2(sum)
{ console.log(sum * 2); }
```

```
funktsiya operationOnSum(num1,num2,operation){
  var sum = num1 + num2;
  operatsiya (sum); }
```

```
OperationOnSum(3, 3, divideByHalf); // Chiqishlar 3
```

```
OperationOnSum(5, 5, multiplyBy2); // Chiqishlar 20
```

- Yuqoridagi kodda biz ikkita sonning yig'indisi bo'yicha matematik amallarni bajaramiz. OperationOnSum funksiyasi 3 ta argumentni, birinchi raqamni, ikkinchi raqamni va ularning yig'indisi (qayta qo'ng'iroq) bo'yicha bajarilishi kerak bo'lgan operatsiyani oladi.
- Ikkala divideByHalf va multiplyBy2 funksiyalari yuqoridagi kodda qayta qo'ng'iroq qilish funksiyalari sifatida ishlataladi.
- Ushbu qayta qo'ng'iroq qilish funksiyalari faqat OperationOnSum funksiyasi bajarilgandan so'ng amalga oshiriladi.
- Shunday qilib, qayta qo'ng'iroq - bu boshqa funktsiya bajarilgandan keyin bajariladigan funktsiya.

24. Javascriptda qanday xatolar turlari mavjud?

Javascriptda ikki xil xatolik mavjud.

1. **Sintaksis xatosi:** Sintaksis xatosi - dasturning umuman bajarilmasligiga yoki yarim yo'lda ishslashni to'xtatib qo'yishiga olib keladigan koddagi xatolar yoki imlo muammolari. Odatda xato xabarlarini ham taqdim etiladi.
2. **Mantiqiy xato:** Sintaksis to'g'ri, lekin mantiq yoki dastur noto'g'ri bo'lса, mulohaza yuritish xatolari yuzaga keladi. Bu holda dastur muammosiz ishlaydi. Biroq, chiqish natijalari noto'g'ri. Ba'zan ularni sintaktik muammolardan ko'ra tuzatish qiyinroq, chunki bu ilovalar mantiqiy nosozliklar uchun xato signallarini ko'rsatmaydi.

25. Memoizatsiya nima?

Memoizatsiya - bu keshlash shakli bo'lib, unda funktsianing qaytish qiymati uning parametrlari asosida keshlanadi. Agar ushbu funktsianing parametri o'zgartirilmasa, funktsianing keshlangan versiyasi qaytariladi.

Keling, oddiy funktsiyani xotirada saqlangan funktsiyaga aylantirish orqali yodlashni tushunaylik:

Eslatma - Yodlash qimmat funktsiya chaqiruvlari uchun ishlatiladi, ammo quyidagi misolda biz yodlash tushunchasini yaxshiroq tushunish uchun oddiy funktsiyani ko'rib chiqamiz.

Quyidagi funktsiyani ko'rib chiqing:

```
funktsiya addTo256(num)
  { qaytish raqami + 256;
    } addTo256(20); // 276 addTo256(40) ni
    qaytaradi; // 296 addTo256(20) ni
    qaytaradi; // 276-ni qaytaradi
```

Yuqoridagi kodda biz parametrni 256 ga qo'shadigan va uni qaytaradigan funksiyanı yozdik.

AddTo256 funksiyasını yana bir xil parametr bilan ("yuqoridagi holatda 20") chaqiranimizda, xuddi shu parametr uchun natijani yana hisoblaymiz.

Natijani bir xil parametr bilan qayta-qayta hisoblash yuqoridagi holatda katta muammo emas, lekin tasavvur qiling-a, agar funksiya og'ir ish qilsa, natijani bir xil parametr bilan qayta-qayta hisoblash isrofgarchilikka olib keladi. vaqt.

Bu yerda memoizatsiya boshlanadi, memoizatsiya yordamida biz hisoblangan natijalarni parametrlar asosida saqlashimiz (keshlashimiz) mumkin. Agar funksiyanı chaqirishda yana bir xil parametr ishlatisa, natijani hisoblash o'rniiga biz to'g'ridan-to'g'ri saqlangan (keshlangan) qiymatni qaytaramiz.

Yuqoridagi addTo256 funksiyasini xotirada saqlangan funksiyaga aylantiramiz:

```
funktsiya memoizedAddTo256()
{ var kesh = {};

qaytish funksiysi(num)
{ if(keshdagi son )
    { console.log("keshlangan qiymat");
    keshni qaytarish [son]

}
else{ kesh[num] = num +
256; keshni qaytarish [num];
}

}
} var memoizedFunc = memoizedAddTo256();

memoizedFunc(20); // Oddiy qaytish
memoizedFunc(20); // Keshlangan qaytish
```

Yuqoridagi kodda memoizedFunc funksiyasını xuddi shu parametr bilan ishga tushirsak, natijani qayta hisoblash o'rniiga u keshlangan natijani qaytaradi.

Eslatma: Yodlashdan foydalanish vaqtini tejashga yordam beradi, lekin biz barcha hisoblangan natijalarni saqlaganimiz uchun xotiraning ko'proq sarflanishiga olib keladi.

26. Dasturlash tilida rekursiya nima?

Rekursiya - bu natijaga kelgunga qadar funktsiyaning o'zini qayta-qayta chaqirishi orqali operatsiyani takrorlash usuli.

```
funktsiya qo'shish(raqam)
{ agar (raqam <= 0) { 0 ni
    qaytarish; }
else
{ raqamni qaytarish + qo'shish (raqam - 1);
}
}
qo'yshish(3) => 3 + qo'yshish(2) 3
+ 2 + qo'yshish(1) 3 + 2
+ 1 + qo'yshish(0) 3 + 2 + 1 + 0
= 6
```

 Rekursiv funksiyaga misol:

Quyidagi funksiya rekursiya yordamida massivdagi barcha elementlarning yig'indisini hisoblab chiqadi:

```
funktsiya computeSum(arr)
{ if(arr.length === 1){ qaytish
arr[0];
}
else{ qaytish arr.pop() + computeSum(arr);
}
}
computeSum([7, 8, 9, 99]); // 123-ni qaytaradi
```

27. Javascriptda konstruktur funksiyasidan qanday foydalaniladi?

Konstruktur funktsiyalari javascriptda ob'ektlar yaratish uchun ishlataladi.

Konstruktur funktsiyalardan qachon foydalanamiz?

Agar biz o'xshash xususiyat va usullarga ega bo'lgan bir nechta ob'ektlarni yaratmoqchi bo'lsak, konstruktor funktsiyalaridan foydalaniladi.

**Eslatma- Konstruktor funktiyasining nomi har doim Paskal yozuvida yozilishi kerak:
har bir so'z bosh harf bilan boshlanishi kerak.**

Misol:

```
funktsiya Shaxs (ismi, yoshi, jinsi){
    this.name = name;
    this.age = yosh;
    this.gender = jins;
}

var person1 = new Person("Vivek", 76, "erkak"); console.log(1-
shaxs);

var person2 = new Person("Courtney", 34, "ayol");
console.log(shaxs2);
```

Yuqoridagi kodda biz Person nomli konstruktor funktiyasini yaratdik. Shaxs tipidagi yangi ob'ekt yaratmoqchi bo'lganimizda, uni new kalit so'zidan foydalanib yaratishimiz kerak:

```
var person3 = new Person("Lilly", 17, "ayol");
```

Yuqoridagi kod qatori Person tipidagi yangi ob'ektni yaratadi. Konstruktor funktsiyalari bizga o'xshash ob'ektlarni guruhlash imkonini beradi.

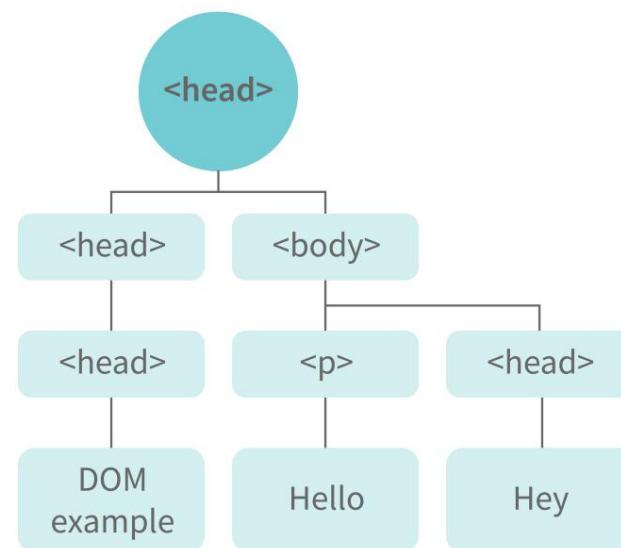
28. DOM nima?

- DOM hujjat ob'ekt modeli degan ma'noni anglatadi. DOM - bu HTML va XML hujjatlari uchun dasturlash interfeysi.
- Brauzer HTML hujjatni ko'rsatishga harakat qilganda, u HTML hujjati asosida DOM deb nomlangan ob'ektni yaratadi. Ushbu DOM-dan foydalanib, biz HTML hujjatidagi turli elementlarni manipulyatsiya qilishimiz yoki o'zgartirishimiz mumkin.
- HTML kodining DOMga qanday o'zgartirilishiga misol:

```

<html>
<head>
<title>
DOM example
</title>
</head>
<body>
<p id =“para1”>Hello</p>
<p id =“para2”>Hey</p>
</body>
</html>

```


InterviewBit


29. Muayyandan belgi olish uchun qaysi usuldan foydalaniladi indeks?

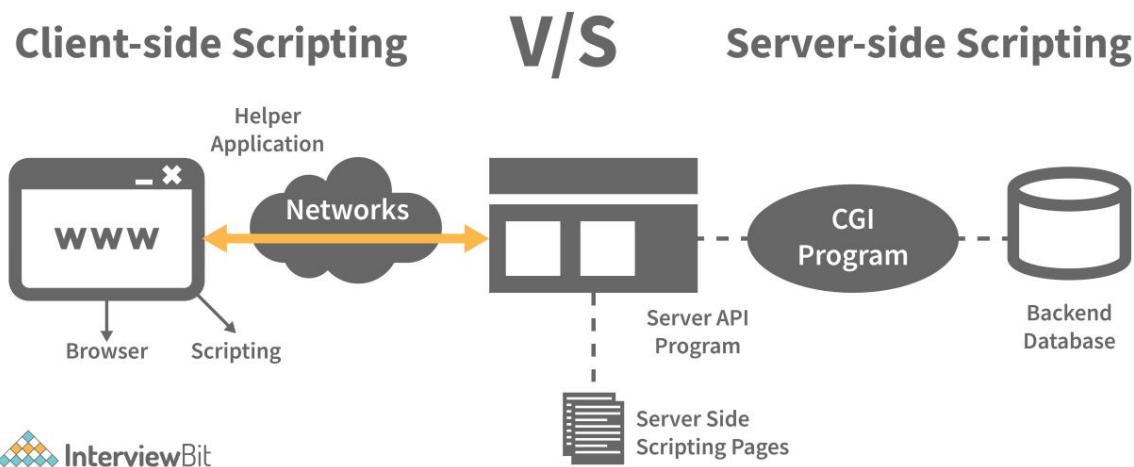
JavaScript qatorining `charAt()` funksiyasi berilgan indeksda char elementini topadi. Indeks raqami 0 dan boshlanadi va n-1 gacha davom etadi, Bu erda n - satr uzunligi. Indeks qiymati musbat, undan yuqori yoki satr uzunligi bilan bir xil bo'lishi kerak.

30. BOM deganda nimani tushunasiz?

Brauzer ob'ekt modeli BOM sifatida tanilgan. Bu foydalanuvchilarga brauzer bilan muloqot qilish imkonini beradi. Brauzerning boshlang'ich obyekti oynadir. Natijada, siz oynaning barcha funktsiyalarini to'g'ridan-to'g'ri yoki oynaga havola qilish orqali chaqirishingiz mumkin. Hujjat, tarix, ekran, navigator, joylashuv va boshqa atributlar oyna obyektida mavjud.

31. Mijoz tomoni va server tomoni JavaScript-ning farqi nimada?

Mijoz tomoni JavaScript ikki qismidan iborat, asosiy til va brauzerda JavaScript-ni bajarish uchun oldindan belgilangan ob'ektlar. Mijoz uchun JavaScript avtomatik ravishda HTML sahifalariga kiritiladi. Ishlash vaqtida brauzer ushbu skriptni tushunadi.



Mijoz tomoni JavaScript server tomonidagi JavaScript-ga o'xshaydi. U serverda bajariladigan JavaScript-ni o'z ichiga oladi. Faqat havoda ishlov berish server tomonidagi JavaScript-ni o'rnatadi.

Tajribali uchun JavaScript Intervyu savollari

32. Ok funksiyalari nima?

Ok funksiyalari javascriptning ES6 versiyasida joriy qilingan. Ular bizga funksiyalarni e'lon qilish uchun yangi va qisqaroq sintaksisni taqdim etadi. Ok funksiyalaridan faqat funksiya ifodasi sifatida foydalanish mumkin.

Oddiy funksiya deklaratsiyasi va o'q funksiysi deklaratsiyasini batafsil taqqoslaylik:

```
// An'anaviy funksiya ifodasi var add =
function(a,b){ return a + b;
```

```
}
```

```
// Ok funksiyasi ifodasi var arrowAdd =
(a,b) => a + b;
```

Ok funksiyalari funksiya kalit so'zisiz e'lon qilinadi. Agar faqat bitta qaytaruvchi ifoda bo'lsa, yuqoridagi misolda ko'rsatilganidek, biz qaytish kalit so'zini o'q funksiyasida ham ishlatishimiz shart emas. Bundan tashqari, faqat bir qator kodga ega funksiyalar uchun {} jingalak qavslar olib tashlanishi mumkin.

```
// Funktsiyaning an'anaviy ifodasi var multiplyBy2
= function(num){ 2;
    raqamni qaytarish
}
// Ok funksiya ifodasi var arrowMultiplyBy2      *
= num => num          2;
```

Agar funksiya faqat bitta argumentni qabul qilsa, yuqoridagi kodda ko'rsatilganidek, parametr atrofidagi qavsniga () olib tashlash mumkin.

```
var obj1 =
{ valueOfThis: function(){ buni
    qaytaring ;
}

} var obj2 =
{ valueOfThis: ()=>{ buni
    qaytaring ;
}
}

obj1.valueOfThis(); // obj1 ob'ektini qaytaradi obj2.valueOfThis(); // Oyna/
global ob'ektini qaytaradi
```

An'anaviy funksiya ifodasi va o'q funktisiyasi o'rtasidagi eng katta farq **bu** kalit so'z bilan ishlashdir . Umumiy ta'rifga ko'ra, **bu** kalit so'z har doim funktisiyani chaqiradigan ob'ektga ishora qiladi. Yuqoridagi kodda ko'rib turganizingizdek, **obj1.valueOfThis()** obj1 ni qaytaradi, chunki **bu** kalit so'z funktisiyani chaqiruvchi ob'ektga ishora qiladi.

O'q funktisiyalarida **bu** kalit so'zning bog'lanishi yo'q . Ok funktisiyasi ichidagi bu kalit so'z uni chaqiruvchi ob'ektga ishora qilmaydi. U o'z qiymatini bu holda oyna ob'ekti bo'lgan ota-onada doirasidan meros qilib oladi. Shuning uchun yuqoridagi kodda **obj2.valueOfThis()** oyna obyektini qaytaradi.

33. Prototip dizayn namunasi deganda nimani tushunish kerak?

Prototip namunasi turli xil ob'ektlarni ishlab chiqaradi, lekin ishga tushirilmagan ob'ektlarni qaytarish o'rniga, shablon yoki namuna ob'ektidan takrorlangan qiymatlarga ega ob'ektlarni ishlab chiqaradi. Prototiplar namunasi sifatida ham tanilgan Prototip namunasi prototiplarni yaratish uchun ishlatiladi.

Ma'lumotlar bazasining standart sozlamalariga mos keladigan parametrlarga ega biznes ob'ektlarining kiritilishi Prototip namunasi qayerda foydali bo'l shining yaxshi namunasidir. Yangi yaratilgan biznes ob'ektining standart sozlamalari prototip ob'ektida saqlanadi.

Prototip namunasi an'anaviy tillarda deyarli qo'llanilmaydi, ammo u prototip tili bo'lgan JavaScript-da yangi ob'ektlar va shablonlarni ishlab chiqishda qo'llaniladi.

34. Var, let va yordamida o'zgaruvchilarni e'lon qilish o'rtasidagi farqlar const.

Javascriptning ES6 versiyasidan oldin o'zgaruvchilarni e'lon qilish uchun faqat var kalit so'zi ishlatilgan. ES6 versiyasi bilan o'zgaruvchilarni e'lon qilish uchun let va const kalit so'zlari kiritildi.

kalit so'z	const	nuxsat bering	var
global qamrov	yo'q	yo'q	ha
funksiya doirasi	ha	ha	ha
blok doirasi	ha	ha	yo'q
qayta tayinlanishi mumkin	yo'q	ha	ha

Keling, misollar bilan farqlarni tushunaylik:

```

var variable1 = 23;
keling o'zgaruvchi2 = 89;

catchValues() funksiyasi
    console.log(variable1);
    console.log(variable2);

// Ikkala o'zgaruvchiga ham istalgan joydan kirish mumkin, chunki ular global scda e'lon qilingan
}

window.variable1; // 23 qiymatini qaytaradi

window.variable2; // Aniqlanmagan qaytaradi

```

- Global miqyosda let kalit so'zi bilan e'lon qilingan o'zgaruvchilar xuddi global miqyosda var kalit so'zi bilan e'lon qilingan o'zgaruvchilar kabi ishlaydi.
- Var va let kalit so'zlari bilan global miqyosda e'lon qilingan o'zgaruvchilarga kodning istalgan joyidan kirish mumkin.
- Biroq, bitta farq bor! Global miqyosda var kalit so'zi bilan e'lon qilingan o'zgaruvchilar oyna/global ob'ektga qo'shiladi. Shuning uchun ularga window.variableName yordamida kirish mumkin.

Holbuki, let kalit so'zi bilan e'lon qilingan o'zgaruvchilar global ob'ektga qo'shilmaydi, shuning uchun window.variableName yordamida bunday o'zgaruvchilarga kirishga urinish xatoga olib keladi.

funktional doirada var vs let

```
function varVsLetFunction(){ let
    awesomeCar1 = "Audi"; var
    awesomeCar2 = "Mercedes"; }

console.log(awesomeCar1); // Xatolik chiqaradi
console.log(awesomeCar2); // Xatolik chiqaradi
```

O'zgaruvchilar **var** yordamida funktional/mahalliy miqyosda e'lon qilinadi va kalit so'zlar aynan bir xil bo'lishiga **imkon beradi**, ya'ni ularga doiradan tashqarida kirish mumkin emas.

```

{ variable3 = [1, 2, 3, 4]; }

console.log(variable3); // Chiqishlar [1,2,3,4]

{ let o'zgaruvchi4 = [6, 55, -1, 2]; }

console.log(variable4); // Otish xatosi

for( i = 0; i < 2; i++){
    // Biror narsa qiling }

console.log(i); // Otish xatosi

for(var j = 0; j < 2; i++){
    // Nimadir qil }

console.log(j) // Chiqishlar 2

```

- Javascriptda blok jingalak qavslar ichida yozilgan kodni bildiradi {}.
- **var** kalit so'zi bilan e'lon qilingan o'zgaruvchilar blok doirasiga ega emas. Bu **var** kalit so'zi bilan {} blok doirasida e'lon qilingan o'zgaruvchi global miqyosda o'zgaruvchini e'lon qilish bilan bir xil ekanligini anglatadi.
- Blok doirasi ichida **let** kalit so'zi bilan e'lon qilingan o'zgaruvchilarga blokdan tashqaridan kirish mumkin emas.

Const kalit so'zi

- **const** kalit so'zi bo'lgan o'zgaruvchilar xuddi let kalit so'zi bilan e'lon qilingan o'zgaruvchiga o'xshab, faqat bitta farq bilan ishlaydi, **const kalit so'zi bilan e'lon qilingan har qanday o'zgaruvchini qayta tayinlab bo'lmaydi.**
- Misol:

```

const x = {ism:"Vivek"};
x = {manzil: "Hindiston"}; // Xatolik chiqaradi
x.name = "Nikhil"; // Hech qanday xatolik yo'q
const y = 23;
y = 44; // Xatolik chiqaradi

```

Yuqoridagi kodda biz **const** kalit so'zi bilan e'lon qilingan o'zgaruvchi ichidagi xususiyat qiymatini o'zgartirishimiz mumkin bo'lisa-da , biz o'zgaruvchining o'zini butunlay qayta tayinlay olmaymiz.

35. Dam olish parametri va tarqalish operatori nima?

Ham dam olish parametri, ham tarqalish operatori javascriptning ES6 versiyasida kiritilgan.



Dam olish parametri (...):

- Bu funksiya parametrlari bilan ishslashning takomillashtirilgan usulini taqdim etadi.
- Qolgan parametr sintaksisidan foydalanib, biz o'zgaruvchan sonli argumentlarni qabul qila oladigan funktsiyalarni yaratishimiz mumkin.
- Har qanday miqdordagi argumentlar rest parametri yordamida massivga aylantiriladi.
- Shuningdek, u argumentlarning barchasini yoki ayrim qismlarini ajratib olishga yordam beradi.
- Dam olish parametrlaridan parametrlardan oldin uchta nuqta (...) qo'yish orqali foydalanish mumkin.

```

function extractingArgs(...args){ args[1]
    ni qaytarish ;
}

// extractingArgs(8,9,1); // 9-ni qaytaradi

funktsiya addAllArgs(...args)
{ sumOfArgs = 0
  bo'lsin ; i =
  0 bo'lsin ; while(i < args.length)
  { sumOfArgs += args[i]; i++;
}

sumOfArgsni qaytarish ;
}

addAllArgs(6, 5, 7, 99); // 117 ta addAllArgs(1, 3, 4)
qaytaradi ; // 8-ni qaytaradi

```

****Izoh - Dam olish parametri har doim funktsiyaning oxirgi parametrida ishlatalishi kerak:**

```

// Dam olish parametri funksiyasidan
foydalanishning noto'g'ri usuli

randomFunc(a,...args,c){ //Biror narsa qiling }

// Dam olish parametri funksiyasidan to'g'ri
foydalanish usuli randomFunc2(a,b,...args){ //
Biror narsa qiling }

```

- **Spread operatori (...):** Spred operatorining sintaksisi qolgan parametr bilan aynan bir xil bo'lsa-da, tarqalish operatori massiv va ob'ekt literallarini tarqatish uchun ishlataladi. Funktsiya chaqiruvida bir yoki bir nechta argumentlar kutiladigan tarqalish operatorlaridan ham foydalanamiz.

```

funktsiya addFourNumbers(num1,num2,num3,num4){ 1-raqam
    + 2-raqam + 3-raqam + 4-raqamni qaytarish ;
}

to'rtNumbers = [5, 6, 7, 8] bo'lisin ;

addFourNumbers(...to'rtNumber); // 
[5,6,7,8] ni 5,6,7,8 sifatida tarqatadi

let massivi1 = [3, 4, 5, 6]; let
clonedArray1 = [...array1]; // Massivni
3,4,5,6 ga tarqatadi
console.log(clonedArray1); // Chiqishlar [3,4,5,6]

let obj1 = {x:'Salom', y:'Xayr'}; let
clonedObj1 = {...obj1}; // Spreadlar va klonlar obj1 console.log(obj1);

let obj2 = {z:'Ha', a:'Yo'}; let
mergedObj = {...obj1, ...obj2}; // Ikkala ob'ektni ham tarqatadi va uni birlashtiradi
console.log(mergedObj); //
Chiqishlar {x:'Salom', y:'Bye',z:'Ha',a:'Yo'};

```

***Izoh - Dam olish parametri va tarqalish operatori o'rtasidagi asosiy farqlar:

- Rest parametri o'zgaruvchan sonli argumentlarni olish va ularni massivga aylantirish uchun ishlatiladi, bunda tarqalish operatori massiv yoki ob'ektni oladi va uni tarqatadi.
-

36. JavaScript-da qancha turli usullarni yaratishingiz mumkin ob'ekt?

JavaScript-da ob'ektni e'lon qilish yoki qurishning bir necha usullari mavjud.

1. Ob'ekt.
2. Class
yordamida. 3. Metod yaratish.
4. Obyekt harflari. 5.
Funktsiyadan foydalanish.
6. Obyekt konstruktori.

37. Javascriptda va'dalar qanday qo'llaniladi?

Va'dalar javascriptda asinxron operatsiyalarni bajarish uchun ishlataladi.

Va'dalardan oldin, asinxron operatsiyalarni bajarish uchun qayta qo'ng'iroqlar ishlataligan. Ammo qayta qo'ng'iroqlarning cheklangan funksionalligi tufayli asinxron kodni qayta ishlash uchun bir nechta qayta qo'ng'iroqlardan foydalanish boshqarib bo'lmaydigan kodga olib kelishi mumkin.

Va'da ob'ekti to'rtta holatga ega -

- Kutilmoqda - va'daning dastlabki holati. Bu holat va'da bajarilmagan yoki rad etilmaganligini bildiradi, u kutilayotgan holatda.
- Bajarildi - Bu holat va'da bajarilganligini bildiradi, ya'ni asinxronizatsiya tugallangan.
- Rad etilgan - Bu holat va'daning ba'zi sabablarga ko'ra rad etilganligini bildiradi, ya'ni asinxronlik operatsiyasi muvaffaqiyatsiz tugadi.
- O'rnatilgan - Bu holat va'da rad etilgan yoki bajarilganligini anglatadi.

Va'da ikki parametr bilan qayta qo'ng'iroq qilish funksiyasini qabul qiluvchi **Promise** konstruktori yordamida yaratiladi , mos ravishda **hal qiladi** va **rad etadi** .

new Promise ()

resolve ()

Go to next action

reject ()

Handle Error



hal qilish - asinxronlash jarayoni muvaffaqiyatli yakunlanganda chaqiriladigan funksiya.



rad etish asinxronizatsiya bajarilmasa yoki xatolik yuzaga kelganda chaqiriladigan funksiyadir yuzaga keladi.

Va'daga misol:

Va'dalar server so'rovlari kabi asinxron operatsiyalarni bajarish uchun ishlatiladi, tushunish qulayligi uchun biz uchta elementning yig'indisini hisoblash uchun operatsiyadan foydalanamiz.

Quyidagi funktsiyada biz funktsiya ichidagi va'dani qaytaramiz:

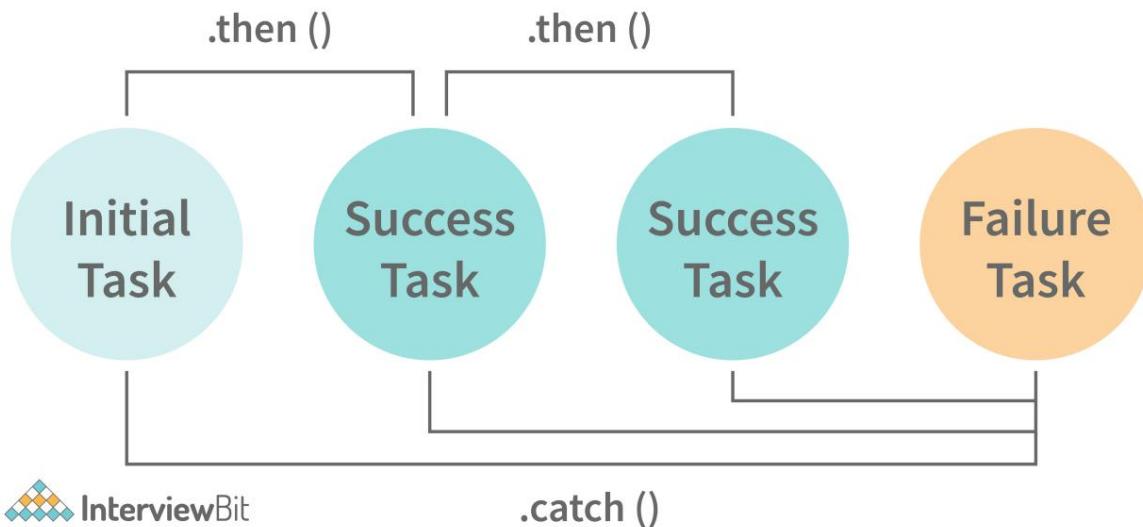
```

funktsiya sumOfThreeElements(...elementlar){ yangi
    va'dani qaytaring((hal qilish, rad etish)=>{
        agar(elementlar.uzunligi > 3 ){
            rad etish ("Faqat uchta yoki undan kam elementga ruxsat beriladi");
        }
        else{ sum = 0; i
            = 0 bo'lsin ;
            while(i < elements.length){ summa
                += elements[i]; i++;
            }
        } rezolyutsiya("summa hisoblandi: "+sum);
    } })
}

```

Yuqoridagi kodda biz uchta elementning yig'indisini hisoblaymiz, agar elementlar massivining uzunligi 3 dan ortiq bo'lisa, va'da rad etiladi yoki va'da hal qilinsa va summa qaytariladi.

Keyin() va catch() usullarini biriktirish orqali har qanday va'dani ishlatsishimiz mumkin iste'molchi.



Keyin() usuli va'da bajarilganda natijaga kirish uchun ishlataladi.

catch() usuli va'da rad etilganda natija/xatoga kirish uchun ishlataladi. Quyidagi kodda biz va'dani ishlataшимиз:

```
sumOfThreeElements(4, 5,  
6) .then(natija=> console.log(natija)) .catch(xato=>  
console.log(xato)); // Yuqoridagi kodda va'da  
bajariladi, shuning uchun then() usuli bajariladi  
  
sumOfThreeElements(7, 0, 33,  
41) .then(natija => console.log(natija)) .catch(xato=>  
console.log(xato)); // Yuqoridagi kodda va'da  
rad etilgan, shuning uchun catch() usuli bajariladi
```

38. Javascriptda qanday sinflar mavjud?

ES6 versiyasida taqdim etilgan sinflar konstruktor funktsiyalari uchun sintaktik shakarlardan boshqa narsa emas. Ular javascriptda konstruktor funksiyalarini e'lon qilishning yangi usulini taqdim etadi. Quyida sinflarning qanday e'lon qilinishi va ishlatalishiga misollar keltirilgan:

```
// ES6 versiyasidan oldin konstruktor funksiyasidan foydalanib
Student (nom, rollNumber, grade, section){
    this.name = name;
    this.rollNumber = rollNumber; this.grade
    = baho; this.section =
    bo'lim;
}

// Konstruktor funksiyasiga usullar qo'yishish usuli
Student.prototype.getDetails = function(){
    Qaytish 'Ism: ${this.name}, Ro'yxat raqami: ${this.rollNumber}, Baho: ${this.grade}, Bo'lim
}

let student1 = new Student("Vivek", 354, "6th", "A");
student1.getDetails(); //
Qaytaradi nomi: Vivek, Roll ý: 354, sinf: 6, bo'lim: A

// ES6 versiya sinflari sinf

Student{ konstruktor(nom,rollNumber,grade,section){ this.name =
    name; this.rollNumber =
    rollNumber; this.grade = baho; this.section =
    bo'lim;
}

// Metodlar to'g'ridan-to'g'ri getDetails() sinfiga qo'shilishi mumkin
{ return 'Name: $  

    {this.name}, Roll no: ${this.rollNumber}, Grade:${this.grade}, Secti
    }
}
}

let student2 = new Student("Garry", 673, "7th", "C");
student2.getDetails(); //
Qaytaradi nomi: Garri, Roll ý: 673, Sinf: 7, Bo'lim: C
```

Sinflar haqida eslash kerak bo'lgan asosiy fikrlar:

- Funktsiyalardan farqli o'laroq, sinflar ko'tarilmaydi. Sinfni e'lon qilishdan oldin ishlatib bo'lmaydi.
- Sinf kengaytirilgan kalit so'z yordamida boshqa sinflardan xususiyatlar va usullarni meros qilib olishi mumkin.
- Sinf ichidagi barcha sintaksislari javascriptning qat'iy rejimiga ("qattiq foydalanish") amal qilishi kerak. Qattiq rejim qoidalariiga rioya qilinmasa, xatolik yuzaga keladi.

39. Generator funktsiyalari nima?

ES6 versiyasida taqdim etilgan generator funktsiyalari funktsiyalarning maxsus sinfidir.

Ularni o'rtada to'xtatib, keyin to'xtagan joydan davom ettirish mumkin.

Generator funktsiyalari normal **funktsiya** kalit so'zi o'rniiga **function*** kalit so'zi bilan e'lon qilinadi :

```
funktsiya* genFunc()
// Amaliyotni bajarish }
```

Oddiy funktsiyalarda biz qiymatni qaytarish uchun **return** kalit so'zidan foydalanamiz va return iborasi bajarilishi bilanoq funktsiyaning bajarilishi to'xtaydi:

```
funktsiya normalFunc()
{ qaytish 22;
console.log(2); // Ushbu kod qatori bajarilmaydi }
```

Jenerator funktsiyalari bo'lsa, chaqirilganda, ular kodni bajarmaydi, aksincha, ular **generator ob'ektini qaytaradi**. Ushbu generator ob'ekti bajarilishini boshqaradi.

```
funktsiya* genFunc(){ hosil
3; hosil
4; }

genFunc(); // Ob'ektni qaytaradi [Generator] {}
```

Generator ob'ekti **next()** deb nomlangan usulidan iborat bo'lib, bu usul chaqirilganda kodni eng yaqin **rentabellik** bayonotigacha bajaradi va rentabellik qiymatini qaytaradi.

Masalan, yuqoridaagi kodda keyingi() usulini ishga tushirsak:

```
genFunc().keyingi(); // Qaytaradi {qiyomat: 3, bajarildi: noto'g'ri}
```

Ko'rib turganingizdek, keyingi usul **qiymat** va **bajarilgan** xususiyatlardan iborat ob'ektni qaytaradi. Value xususiyati olingan qiymatni ifodalaydi. Done xususiyati funktsiya kodi tugagan yoki tugallanmaganligini bildiradi. (Agar tugallangan bo'lsa, true qiymatini qaytaradi).

Generator funktsiyalari iteratorlarni qaytarish uchun ishlataladi. Keling, iterator qaytarilgan misolni ko'rib chiqaylik:

```
function* iteratorFunc() { count =
  0; uchun ( i = 0; i
  < 2; i++) { count++; hosil i;

} qaytish soni;
}

let iterator = iteratorFunc();
console.log(iterator.next()); // {qiymat:0,bajarildi:yolg'on}
console.log(iterator.next()); // {qiymat: 1, bajarildi: noto'g'ri}
console.log(iterator.next()); // {qiymat: 2, bajarildi: rost}
```

Yuqoridagi kodda ko'rib turganingizdek, oxirgi satr **bajarilgan: rostni qaytaradi**, chunki kod qaytish bayonotiga etib boradi.

40. Javascriptda WeakSet ni tushuntiring.

Javascriptda to'plam noyob va tartiblangan elementlar to'plamidir. Xuddi Set singari, WeakSet ham ba'zi muhim farqlarga ega noyob va tartiblangan elementlar to'plamidir:

- Zaif to'plam faqat ob'ektlarni o'z ichiga oladi va boshqa turlar yo'q.
- Zaif to'plam ichidagi ob'ektga zaif havola qilinadi. Bu shuni anglatadiki, agar zaif to'plam ichidagi ob'ektda mos yozuvlar bo'lmasa, u axlat yig'iladi.
- Setdan farqli o'laroq, WeakSet faqat uchta usulga ega, **add()** , **delete()** va **has()** .

```

const newSet = yangi to'plam([4, 5, 6, 7]);
console.log(newSet); // Chiqishlar to'plami {4,5,6,7}

const newSet2 = new WeakSet([3, 4, 5]); // Xatoga yo'l qo'ydi

let obj1 = {xabar:"Salom dunyo"};
const
newSet3 = new WeakSet([obj1]);
console.log(newSet3.has(obj1)); // rost

```

41. Nima uchun biz qayta qo'ng'iroqlardan foydalanamiz?

Qayta qo'ng'iroq funktsiyasi boshqa funktsiyaga kirish sifatida yuboriladigan usuldir (endi bu boshqa funktsiyani "thisFunction" deb nomlaymiz) va u funktsiya bajarilishini tugatgandan so'ng thisFunction ichida amalga oshiriladi.

JavaScript voqealarga asoslangan skript tilidir. Davom etishdan oldin javobni kutish o'rniiga, JavaScript qo'shimcha hodisalarini kuzatishda ishlashda davom etadi. Qayta qo'ng'iroqlar - bu ma'lum bir kod boshqa kod bajarilishini tugatmaguncha ishlamasligini ta'minlash usuli.

42. WeakMap ni javascriptda tushuntiring.

Javascriptda Map kalit-qiyomat juftlarini saqlash uchun ishlatiladi. Kalit-qiyomat juftlari ham ibtidoiy, ham ibtidoiy bo'limgan turdag'i bo'lishi mumkin. WeakMap asosiy farqlari bilan Mapga o'xshaydi:

- Zaif xaritadagi kalitlar va qiyamatlar har doim ob'ekt bo'lishi kerak.
- Agar ob'ektga havolalar bo'limasa, ob'ekt axlat yig'iladi.

```

const map1 = new Map();
map1.set('Qiymat', 1);

const map2 = new WeakMap();
map2.set('Qiymat', 2.3); // Xatolik chiqaradi

let obj = {ism:"Vivek"};
const
map3 = new WeakMap();
map3.set(obj, {yosh:23});

```

43. Obyekt tuzilmasini buzish nima?

Ob'ektni buzish - bu ob'ekt yoki massivdan elementlarni ajratib olishning yangi usuli.

- **Ob'ektni buzish:** ES6 versiyasidan oldin:

```
const classDetails = {kuch: 78,
    skameykalar: 39,
    qora taxta:1

}

const classStrength = classDetails.strength; const classBenches =
classDetails.benches; const classBlackBoard = classDetails.blackBoard;
```

Ob'ektni buzishdan foydalangan holda xuddi shu misol:

```
const classDetails = {kuch: 78,
    skameykalar: 39,
    qora taxta:1

}

const {kuch:classStrength, skameykalar:classSkameykalar,blackBoard:classBlackBoard} = sif

console.log(classStrength); // 78 ta console.log (classBenches); //
39 chiqdi console.log(classBlackBoard); // Chiqishlar 1
```

Ko'rinib turibdiki, ob'ektni destruksiyalashdan foydalanib, biz kodning bir qatorida ob'ekt ichidagi barcha elementlarni chiqarib oldik. Agar biz yangi o'zgaruvchimiz ob'ektning xususiyati bilan bir xil nomga ega bo'lishini istasak, biz ikki nuqtani olib tashlashimiz mumkin:

```
const {kuch: kuch} = classDetails;
// Yuqoridagi kod qatori quyidagicha yozilishi mumkin: const {strength}
= classDetails;
```

- **Massivni buzish:** ES6 versiyasidan oldin:

```
const arr = [1, 2, 3, 4]; const first =
arr[0]; const second = arr[1];
const uchinchi = arr[2]; const
to'rtinchi = arr[3];
```

Ob'ektni buzishdan foydalangan holda xuddi shu misol:

```
const arr = [1, 2, 3, 4]; const
[birinchi, ikkinchi, uchinchi, to'rtinchi] = arr;
console.log(birinchi); // 1 ta console.log
(ikkinchi) chiqadi ; // 2 ta console.log
(uchinchi); // 3 ta console.log
chiqdi(to'rtinchi); // Chiqishlar 4
```

44. Prototip va klassik merosning farqi

Dasturchilar an'anaviy OO dasturlashda real vaqtda ob'ektlarning tasviri bo'lgan ob'ektlarni quradilar. Sinflar va ob'ektlar mavhumlikning ikki turidir. Sinf - bu ob'ektning umumlashtirilishi, ob'ekt esa haqiqiy narsaning mavhumligi. Masalan, avtomashina bu avtomobilning ixtisoslashuvidir. Natijada, avtobillar (sinf) transport vositalaridan (ob'ektdan) kelib chiqadi.

Klassik meros prototip merosdan farq qiladi, chunki klassik meros qolgan sinflardan meros bo'lgan sinflar bilan chegaralanadi, ammo prototip meros har qanday ob'ektni ob'ektni bog'lash usuli orqali klonlash imkonini beradi. Juda ko'p o'ziga xos xususiyatlarga ega bo'lishiga qaramay, prototip asosiy ob'ektni kengaytiradimi yoki yo'qmi, boshqa ob'ektlar uchun shablon bo'lib xizmat qiladi.

45. Vaqtinchalik o'lik zona nima?

Vaqtinchalik o'lik zona - bu **let** va **const** kalit so'zlari yordamida e'lon qilingan o'zgaruvchilar bilan sodir bo'ladigan xatti-harakatlar . Bu o'zgaruvchiga uni ishga tushirishdan oldin kirishga harakat qiladigan xatti-harakatlardir. Vaqtinchalik o'lik zonaga misollar:

```
x = 23; // Malumot xatosi beradi

x bo'lsin ;

function anotherRandomFunc()
{ message = "Salom"; // Malumot xatosini chiqaradi

  xabar bering ;

anotherRandomFunc();
```

Yuqoridagi kodda ham global miqyosda, ham funksional doirada biz hali e'lon qilinmagan o'zgaruvchilarga kirishga harakat qilmoqdamiz. Bu vaqtinchalik o'lik zona deb ataladi .

46. JavaScript Design Patterns deganda nimani tushunasiz?

JavaScript dizayn naqshlari ba'zan JavaScript brauzer ilovalarini yaratishda yuzaga keladigan xatolar uchun takrorlanadigan yondashuvlardir. Ular bizning kodimizni yanada barqaror qilishda bizga yordam beradi.

Ular asosan 3 toifaga bo'lingan

1. Yaratuvchi dizayn namunasi
2. Strukturaviy dizayn namunasi
3. Behavioral dizayn namunasi.

- **Creational Design Pattern:** Ob'ektni yaratish mexanizmi JavaScript Creational Design Pattern tomonidan ko'rib chiqiladi. Ular ma'lum bir stsenariyiga mos keladigan narsalarni yaratishga intiladi.
- **Strukturaviy dizayn namunasi:** JavaScript Strukturaviy dizayn namunasi biz hozirgacha yaratgan sinflar va ob'ektlarni qanday qilib kattaroq ramkalar yaratish uchun birlashtirish mumkinligini tushuntiradi. Ushbu naqsh ob'ektlar o'rtasidagi munosabatlarni yaratishning to'g'ridan-to'g'ri usulini belgilash orqali osonlashtiradi.
- **Xulq-atvor dizayni namunasi:** Ushbu dizayn namunasi JavaScript-dagi ob'ektlar o'rtasidagi aloqaning odatiy modellarini ta'kidlaydi. Natijada, aloqa ko'proq erkinlik bilan amalga oshirilishi mumkin.

47. JavaScript o'tish-by-reference yoki pass-by-value tilimi?

O'zgaruvchining ma'lumotlari har doim ob'ektlar uchun havola hisoblanadi, shuning uchun u har doim qiymat bo'yicha o'tadi. Natijada, agar siz ob'ektni taqdim qilsangiz va uning a'zolarini usul ichida o'zgartirsangiz, o'zgarishlar undan tashqarida davom etadi. Bu holatda havola orqali o'tganga o'xshaydi. Biroq, agar siz ob'ekt o'zgaruvchisining qiymatlarini o'zgartirsangiz, o'zgarish davom etmaydi, bu haqiqatan ham qiymat bo'yicha uzatilganligini ko'rsatadi.

48. Async/Await va Generatorlardan foydalanish o'rtasidagi farq bir xil funksionallikka erishing.

- Generator funksiyalari ishlab chiqaruvchining rentabelligi bo'yicha ishlab chiqariladi, bu bir vaqtning o'zida bitta chiqishni bildiradi, async-await funksiyalari esa ketma-ket bajariladi.
- Async/await Generatorlar uchun ma'lum bir foydalanish holatini amalga oshirishni osonlashtiradi.
- Generator funksiyasining chiqish natijasi har doim qiymat: X, bajarildi: Mantiqiy, lekin Async funksiyasining qaytish qiymati har doim kafolatdir yoki xato.

49. JavaScript-da qanday primitiv ma'lumotlar turlari mavjud?

Primitiv - bu boshqa ma'lumotlar turlaridan iborat bo'limgan ma'lumotlar turi. U bir vaqtning o'zida faqat bitta qiymatni ko'rsatishga qodir. Ta'rifga ko'ra, har bir ibtidoiy o'rnatilgan ma'lumotlar turidir (kompilyator ular haqida bilishi kerak), ammo barcha o'rnatilgan ma'lumotlar to'plamlari primitiv emas. JavaScript-da asosiy ma'lumotlarning 5 xil shakli mavjud. Quyidagi qiymatlar mavjud:

1. Mantiqiy
 2. Aniqlanmagan
 3. Null
 4. Raqam 5.
- String

50. JavaScript-da kechiktirilgan skriptlarning roli qanday?

Sahifani yuklash paytida HTML kodini qayta ishslash, skript to'xtatilmaguncha, tabiatan o'chirib qo'yilgan. Tarmog'ingiz biroz sekin bo'lsa yoki skript juda yaxshi bo'lsa, sahifangizga ta'sir qiladi. Kechiktirilgandan foydalansangiz, skript HTML tahlilchisi uni bajarishdan oldin tugashini kutadi. Bu veb-sahifalarni yuklash uchun ketadigan vaqtini qisqartiradi va ularning tezroq paydo bo'lishiga imkon beradi.

51. Leksik qamrovni amalda qo'llash uchun nima qilish kerak?

Leksik qamrovni qo'llab-quvvatlash uchun JavaScript funksiyasi ob'ektining ichki holati nafaqat funksiya kodini, balki joriy qamrov zanjiriga havolani ham o'z ichiga olishi kerak.

52. Quyidagi JavaScript kodining maqsadi nima?

```
var scope = "global qamrov";
funktsiyani
tekshirish() {
    var scope = "mahalliy qamrov"; f()
    funktsiyasi

    qaytarish doirasi;

} qaytish f;
}
```

JavaScript-dagi har bir bajaruvchi funksiya, kod bloki va umuman skript leksik muhit deb nomlanuvchi tegishli obyektga ega. Oldingi kod satri qamrovdagi qiymatni qaytaradi.

JavaScript kodlash intervyu savollari

53. Quyidagi kodlarning chiqishini taxmin qiling:

// Kod 1:

```
func1 ()

{ setTimeout(()=>{ console.log(x); console.log(y); },3000);

var x = 2; y =
12 bo'lisin ; }
```

func1();

// Kod 2:

```
func2 (){ for(var i = 0; i < 3; i++) { setTimeout(()=>
    console.log(i),2000);
```

} } func2();

// Kod 3:

```
(function()
{ setTimeout(()=> console.log(1),2000);
  console.log(2);
  setTimeout(()=> console.log(3),0);
  console.log(4) } )();
```

Javoblar:

- **Kod 1** - Chiqish **2** va **12. Let** o'zgaruvchilari ko'tarilmasa ham , JavaScript asink xususiyati tufayli, to'liq funktsiya kodi setTimeout funksiyasidan oldin ishlaydi. Shuning uchun u x va y ga ham kirish huquqiga ega.
- **Kod 2** - 3, uch marta chiqadi , chunki **var** kalit so'zi bilan e'lon qilingan o'zgaruvchi blok doirasiga ega emas. Bundan tashqari, for tsikli ichida i o'zgaruvchisi avval ko'paytiriladi va keyin tekshiriladi.
- **Kod 3** - Quyidagi tartibda chiqish:

2
4
3
1 // Ikki soniyadan keyin

Ikkinci vaqt tugashi funksiyasi nol soniya kutish vaqtiga ega bo'lsa ham, JavaScript mexanizmi har doim Web API yordamida setTimeout funksiyasini baholaydi va shuning uchun setTimeout funksiyasi bajarilgunga qadar to'liq funksiya bajariladi.

54. Quyidagi kodning natijalarini taxmin qiling:

// Kod 1:

```
x= {}, y = { ism:"Ronni"},z = {ism:"Jon"}; x[y] = {ism:"Vivek"}; x[z] = {ism:"Akki"};
console.log(x[y]);
```

// Kod 2:

```
runFunc() funksiyasi
console.log("1" + 1); console.log("A"
- 1); console.log(2 + "-2" + "2");
console.log("Salom" - "Dunyo" + 78);
console.log("Salom"+ "78"); } runFunc();
```

// Kod 3:

```
a = 0 bo'lsin ;
b = yolg'on ;
console.log((a == b)); console.log((a
==== b));
```

Javoblar:

Kod 1 - Chiqish {ism: "Akki"} bo'ladi .

Ob'ektlarni boshqa ob'ektning xususiyatlari sifatida qo'shish ehtiyyotkorlik bilan bajarilishi kerak.

Yozish `x[y] = {ism:"Vivek"} , x['object Object'] = yozish bilan bir xil {ism:"Vivek"} ,`

Ob'ektning xususiyatini o'rnatishda **JavaScript parametrni satrga majburlaydi**.

Shuning uchun, `y` ob'ekt bo'lganligi sababli, u "**ob'ekt ob'ekti**" ga aylantiriladi .

Ikkala `x[y]` va `x[z]` bir xil xususiyatga ishora qiladi.

Kod 2 - quyidagi tartibda chiqadi:

```
11
Nan
2-22
NaN
Salom 78
```

Kod 3 - Tenglikka majburlash tufayli quyidagi tartibda chiqish:

```
haqiqiy
yolg'on
```

55. Quyidagi kodning chiqishini taxmin qiling:

```
var x = 23;

(function(){ var x
= 43; (funktsiya
random(){ x++; console.log(x);
var x
= 21; })(); })();
```

Javob:**Chiqish - NaN.**

random() funktsiyasi funksional doiraga ega, chunki x funksional doirada e'lon qilingan va ko'tarilgan.

Tasodifiy funktsiyanı qayta yozish chiqish haqida yaxshiroq fikr beradi:

```
funktsiya random(){ var x; //
x ko'tarildi x++; // x bu raqam emas,
chunki u hali ishga tushirilmagan console.log(x); // Chiqishlar NaN x = 21; // x ni ishga
tushirish }
```

56. Quyidagi kodning chiqishlarini taxmin qiling:

// Kod 1

```
let hero =
  { powerLevel: 99,
    getPower()
      { this.powerLevelni qaytaring;
    }
  }

let getPower = hero.getPower;

let hero2 = {powerLevel:42};
console.log(getPower());
console.log(getPower.apply(hero2));
```

// Kod 2

```
const a = function()
  { console.log(bu);

  const b =
    { func1: function(){
      console.log(bu); }

  }

  const c =
    { func2:
      ()=>{ console.log(bu); }

  }

  b.func1();
  c.func2(); }

a();
```

// Kod 3

```
const b =
  { name:"Vivek",
    f: function(){ var
      self = this;
      console.log(tush.name);
      (function()
        { console.log(this.name);
          console.log(self.name); })(); }

  }

} bf();
```

Javoblar:

Kod 1 - Quyidagi tartibda chiqish:

```
aniqlanmagan
42
```

Sabab - Birinchi chiqish **aniqlanmagan** , chunki funktsiya chaqirilganda u global ob'ektga murojaat qilib chaqiriladi:

```
window.getPower() = getPower();
```

Kod 2 - quyidagi tartibda chiqadi:

```
global/oyna ob'ekti "b" global/
oyna ob'ekti
```

Func2 ichidagi strelka funksiyasidan foydalanayotganimiz sababli , bu kalit so'z global obyektga ishora qiladi.

Kod 3 - Quyidagi tartibda chiqadi:

```
"Vivek"
aniqlanmagan
"Vivek"
```

Faqat f funksiyasi ichidagi IIFE-da **bu** kalit so'z global/oyna ob'ektiga ishora qiladi.

57. Quyidagi kodning natijalarini taxmin qiling:

**Izoh - Kod 2 va Kod 3 chiqishni taxmin qilish o'rniga kodni o'zgartirishingizni talab qiladi.

```
// Kod 1
```

```
(funksiya(a)
  { qaytish (funksiya()
    { console.log(a); a
      = 23; })
  () }
(45);
```

```
// Kod 2
```

// Har safar bigFunc chaqirliganda 700 o'lchamdagи massiv yaratiladi,
// Bir xil massivni qayta-qayta yaratmasligimiz uchun kodni o'zgartiring

```
funktsiya bigFunc(element){ let
  newArray = new Array(700).fill('ÿ'); newArray[element] ni
  qaytarish ;
}
```

```
console.log(bigFunc(599)); // Massiv yaratildi
console.log(bigFunc(670)); // Massiv yana yaratildi
```

```
// Kod 3
```

// Quyidagi kod bir soniya kutgandan so'ng 2 va 2 ni chiqaradi // Kodni bir soniyadan keyin 0 va 1 chiqishiga o'zgartiring.

```
funksiya randomFunc()
{ for(var i = 0; i < 2; i++){ setTimeout(()=>
  console.log(i),1000);
}
} randomFunc();
```

Javoblar -

Kod 1 - Chiqishlar 45.

Tashqi funktsiyada a aniqlangan bo'lsa ham, yopilish tufayli ichki funktsiyalar unga kirish huquqiga ega.

Kod 2 - Ushbu kodni yopishlar yordamida o'zgartirish mumkin,

bigFunc() funksiyasi

```
let newArray = new Array(700).fill('y'); return
(element) => newArray[element];
}

let getElement = bigFunc(); // Massiv faqat bir marta yaratiladi
getElement(599);
getElement (670);
```

Kod 3 - ikki yo'l bilan o'zgartirilishi mumkin:

Let kalit so'zidan foydalanish :

funksiya **randomFunc()**

```
{ for(let i = 0; i < 2; i++)
  { setTimeout(()=> console.log(i),1000);
  }

} randomFunc();
```

Yopishdan foydalanish:

funksiya **randomFunc()**

```
{ for(var i = 0; i < 2; i++){ (funksiya(i)

{ setTimeout(()=>console.log(i),1000); })(i);

}

} randomFunc();
```

58. Saralanganda ikkilik qidiruvni bajaradigan funksiyani yozing massiv.

```

funktsiya binarySearch(arr,value,startPos,endPos){
    if(startPos > endPos) qaytish -1;
    let middleIndex = Math.floor(startPos+endPos)/2;
    if(arr[middleIndex] === qiymat) middleIndexni qaytaradi ;
    elsif(arr[middleIndex] > qiymat){ qaytish
        binarySearch(arr,value,startPos,middleIndex-1);
    }
    else{ return binarySearch(arr,value,middleIndex+1,endPos);
    }
}

```

59. a butun sonlar massivida r o'ngga aylanish bilan yangilangan massivni qaytaruvchi funksiyani bajaring.

Misol:

Quyidagi massiv berilgan: **[2,3,4,5,7]**

3 ta o'ngga aylanishni bajaring :

Birinchi aylanish: [7,2,3,4,5], Ikkinci aylanish: [5,7,2,3,4] va, Uchinchi aylanish: [4,5,7,2,3]

qaytish **[4,5,7,2,3]**

Javob:

```

funktsiya rotateRight(arr,rotations){ if(rotations == 0)
    qaytish arr; for(let i = 0; i < rotations;i++)
    { let element = arr.pop(); arr.unshift(element); }
    return arr;
}

} rotateRight([2, 3, 4, 5, 7], 3); // Qaytish [4,5,7,2,3] rotateRight([44, 1, 22,
111], 5); // Qaytadi [111,44,1,22]

```

60. Yangi komponentlarni dinamik kiritish kodini yozing.

```
<html>
<head>
<title>yangi komponentlarni dinamik ravishda kiritish</title> <script type="text/
javascript">
funktsiya addNode () { var newP = hujat. createElement("p"); var textNode = document.createTextNode("Bu
boshqa tugun"); newP.appendChild(textNode); document.getElementById("ota-
ona1").appendChild(newP); } </script> </head> <body> <p id="parent1">birinchiP<p> </body> </html>
```

61. Berilgan kodni yozing Agar ikkita satr bir-birining anagrammasi bo'lsa, u holda true qiymatini qaytaring.

```
var firstWord = "Deepak"; var secondWord
= "Aman";

isAnagram(wordOne, wordTwo); // rost

function isAnagram(bir, ikki) { //Har ikkala so'zni
    kichik harflarga o'zgartiring.. var a = one.toLowerCase(); var b = two.toLowerCase();

    // Satrlarni tartiblang, so'ngra massivni satrga birlashtiring. Natijalarni ko'rib chiqing. a = a.split("").sort().join(""); b =
    b.split("").sort().join("");

    a === b qaytarish ;
}
```

62. Unli tovushlarni topish kodini yozing

```
const findVowels = str => {
  let count = 0 const
  unlilar = ['a', 'e', 'i', 'o', 'u'] for(let char of str.toLowerCase())
  { if(vowels.includes(char)) { hisoblash ++
    }
  }
  qaytish soni
}
```

63. JavaScript-da qanday qilib Ob'ektni Massiv [] ga aylantirasiz?

```
let obj = { id: "1", ism: "user22", yosh: "26", ish: "dasturchi" };

//1-usul: Kalitlarni massivga aylantirish - Object.keys()
console.log(Object.keys(obj)); // ["id",
"ism", "yosh", "ish"]

// 2-usul Qiymatlarni massivga o'zgartiradi - Object.values()
console.log(Object.values(obj)); // ["1",
"user22r", "26", "dasturchi"]

// 3-usul Kalit va qiymatlarni - Object.entries() console.log(Object.entries(obj)); //[[{"id",
"1"}, {"name", "user22"}, {"yosh", "26"}, {"ish",
"dasturchi"}]]
```

64. Quyidagi kodning chiqishi nima?

```
const b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

uchun ( i = 0; i < 10; i++) { setTimeout(() 
  => console.log(b[i]), 1000);
}

for (var i = 0; i < 10; i++) { setTimeout(() 
  => console.log(b[i]), 1000);
}
```

Javob.

```
1  
2 3 4 5 6 7  
8 9 10  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined
```

Xulosa



JavaScript, CSS va HTML-ni alohida "javascript" fayllarida saqlash afzalroqdir. Kod va HTML bo'limlarini ajratish ularni tushunish va ishlashni osonlashtiradi. Ushbu strategiya bir vaqtning o'zida bir nechta dasturchilar uchun ham soddadir. JavaScript kodini yangilash oson. Ko'p sahifalar bir xil JavaScript kodlari guruhidan foydalanishi mumkin. Agar biz tashqi JavaScript skriptlaridan foydalansak va kodni o'zgartirishimiz kerak bo'lса, biz buni bir marta qilishimiz kerak. Shunday qilib, biz raqamdan foydalanishimiz va uni ancha oson saqlashimiz mumkin. Esda tutingki, kasbiy tajriba va tajriba yollashning faqat bir jihatи. Oldingi tajriba va shaxsiy ko'nikmalar qo'nishda (yoki ish uchun ideal arizachini topishda) juda muhimdir.

Esda tutingki, ko'pgina JavaScript tuzilgan intervular bepul va hech qanday to'g'ri javob yo'q. Suhbatdoshlar javobni eslab qolgan bo'lsangiz emas, balki nima uchun shunday javob berganingizni bilishni xohlashadi. Javob berish jarayonini tushuntiring va uni hal qilishga tayyor bo'ling.

Tavsiya etilgan manbalar

- <https://www.interviewbit.com/javascript-cheat-sheet/>
- <https://www.interviewbit.com/online-javascript-compiler/>
- <https://www.interviewbit.com/blog/javascript-features/>
- <https://www.interviewbit.com/javascript-mcq/>
- <https://www.interviewbit.com/blog/javascript-projects/>
- <https://www.interviewbit.com/blog/javascript-ide/>
- <https://www.interviewbit.com/es6-interview-questions/>
- <https://www.interviewbit.com/blog/best-javascript-books/>
- <https://www.interviewbit.com/node-js-interview-questions/>

Intervyu qo'llanmalari

- <https://www.interviewbit.com/technical-interview-questions/>
- <https://www.interviewbit.com/coding-interview-questions/>
- <https://www.interviewbit.com/mock-interview/>
- <https://www.interviewbit.com/blog/>

Qo'shimcha intervju uchun havolalar

Savollar

C Intervyu savollari

Web Api intervju
Savollar

Cpp intervju savollari

Mashina o'rganish bo'yicha intervju
Savollar

Css intervju savollari

Django intervju savollari Dot Net intervju savollari Kubernetes intervju
Savollar

Operatsion tizim bilan suhbat
Savollar

Git intervju savollari

Dbms intervju savollari

Pi Sql intervju savollari

Ansible intervju savollari Java intervju savollari

Php intervju savollari

Kutish rejimidagi intervju
Savollar

Voy, intervju savollari

Docker intervju savollari Mysql intervju savollari

Laravel intervju savollari Asp Net intervju savollari

React Native Interview
Savollar

Java 8 intervju savollari Mongodb intervju
Savollar

Spring Boot intervju
Savollar

Tableau intervju
Savollar

Jenkins intervju savollari

C Sharp intervju savollari

Node Js intervju savollari

Devops intervju savollari

Aws intervju savollari

Linux intervju savollari