ADVANCED JAVASCRIPT CONCEPTS CHEAT SHEET

Zero To Mastery

v 10

HEEELLLOOOOO!

Men Andrei Neagoie, Zero To Mastery Akademiyasining Asoschisi va Bosh O'qituvchisi,man.

Biz boshlang'ich JavaScript o'quvchilari uchun bu JavaScript Cheat Sheetni yaratdik, bu yerda ular modern, ilg'or JavaScript amaliyotlarini o'rganish va JavaScript dasturchilarning eng yaxshi 10%iga aylanishlari mumkin. Endi biz bu bilan barcha veb-dasturchilarga oddiy JavaScript kontseptlarini o'rganish va eslab qolishda yordam beramiz.

Agar siz JavaScriptni o'rganishga boshlasangiz, tabriklaymiz! Veb-dasturchi bo'lish ajoyib kasb-opportuniteti va JavaScript dasturchilariga so'rovlar katta. Veb-dasturchi bo'lish uchun hammasini o'rganish uchun mening To'liq Veb-dasturchi Kodlashtirish Bootcamp kursimni tekshiring.

Hali veb-dasturchi, lekin junior yoki o'rta darajadagi lavozimda qotilganmisiz? Biz 30 kun ichida Sizni Senior Javascript Dasturchilarining bilimlariga tezlashtirib berishimiz mumkin. Bizning JavaScript: The Advanced Concepts kursi tugagandan so'ng, siz JavaScript Dasturchilarining eng yaxshi 10%ida bo'lasiz.

O'qingni bepul boshlang , yuqori darslar bo'limidagi kurs havolalariga o'tib PREVIEW tugmasini bosing.

Xush kelibsiz!
Andrei

Zero To Mastery ning asoschisi va bosh o'qituvchisi,
Andrei Neagoie

P.S. Men hozirgi vaqtda Principles For Programmers nomli kitobni ham yozdim. Uning birinchi besh bobini bepul yuklab olishingiz mumkin .

ZTM
Veb-saytga havola

KREDİTLAR

JavaScript: The Advanced Concepts kursini o'qiyotgan Zero To Mastery talabasi va y-ulduzi, Bri^ney ,ning eslatmalari uchun katta rahmat va kredit bor. Bri^neyning b-oshqa ajablanarli eslatmalari haqida ham ma'lumot olishingiz mumkin .

MUNDARIJA

Leksik muhit

Doira zanjiri

Funksiya va blok doiras

IIFE - Biroq shu paytda chaqirilgan funksiya

Ifoda

Bu

Leksik va dinamik doira

Call, Apply, Bind

Call, **Apply**, **Bind**, **Bind bilan currying**

JavaScript turlari

JavaScriptda obyektlar , Asosiy va non-asosiy turlar , Turlash , Statik va dinamik turlash

Ikki asos: Closures va Prototiplar

Funksiya konstruktori , Prototip orqali meroslash , Prototip vs proto , Chaqlanadigan obyekt , Yuqori darajadagi funksiyalar , Closures , Xotira samaradorligi , Qamrab oluvchi

Ob'yektga yo'naltirilgan dasturlash

Ob'yektga yo'naltirilgan dasturlash,Fabrika Funksiyalar,Do'konlar,Object.create,Constructor funksiyalar,Klass,Maxfiy va ommaviy maydonlar

OOPning 4 asosi

Funksional dasturlash

Toza funksiyalar , Referensial transparensiya , Idempotensiya , Imperativ vs Deklarativ , O'zgarmaslik , Qisman qo'llash , Bor va birlashtirish , Arity

Kompozitsiya vs Miras

OOP muammolari,Yakun

JavaScriptda modullar

Modul patternlari,Modullar bilan muammo,ES6 modullari

Xatolar bilan ishlash

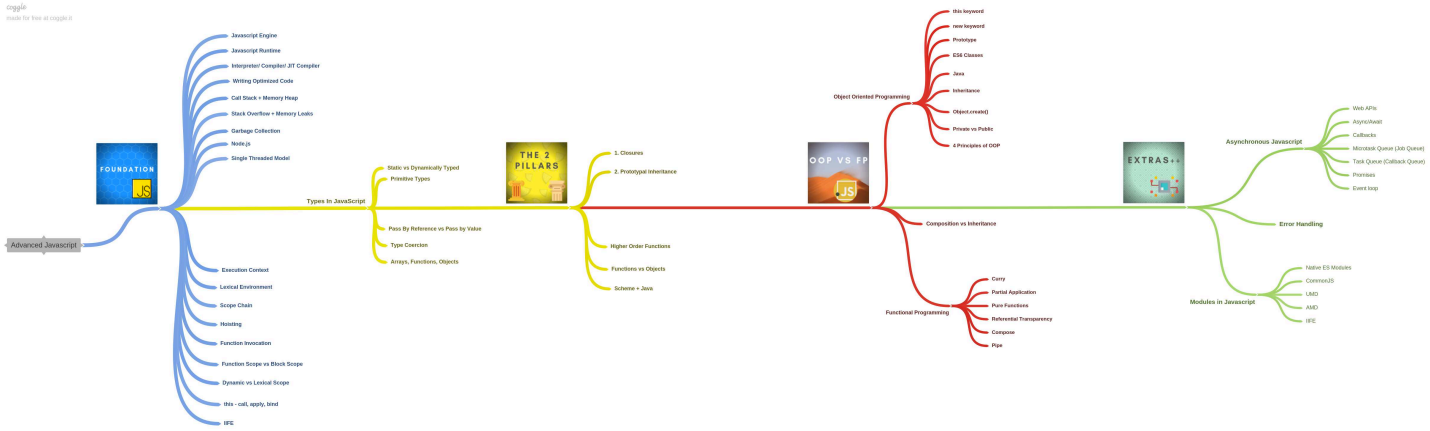
Tugatish...

Ma'lumotlar tuzilishi va algoritmlar

Ma'lumotlar tuzilishi , Kompyuterlar qanday ishlaydi? , Mashhur ma'lumotlar tuzilishlari , Massivlar , Massivni amalga oshirish , Hash jadvali , Hash to'qimlari , JavaScriptda hashlash , Hash jadvali yaratish

Kreditlar

KURS XARITASI

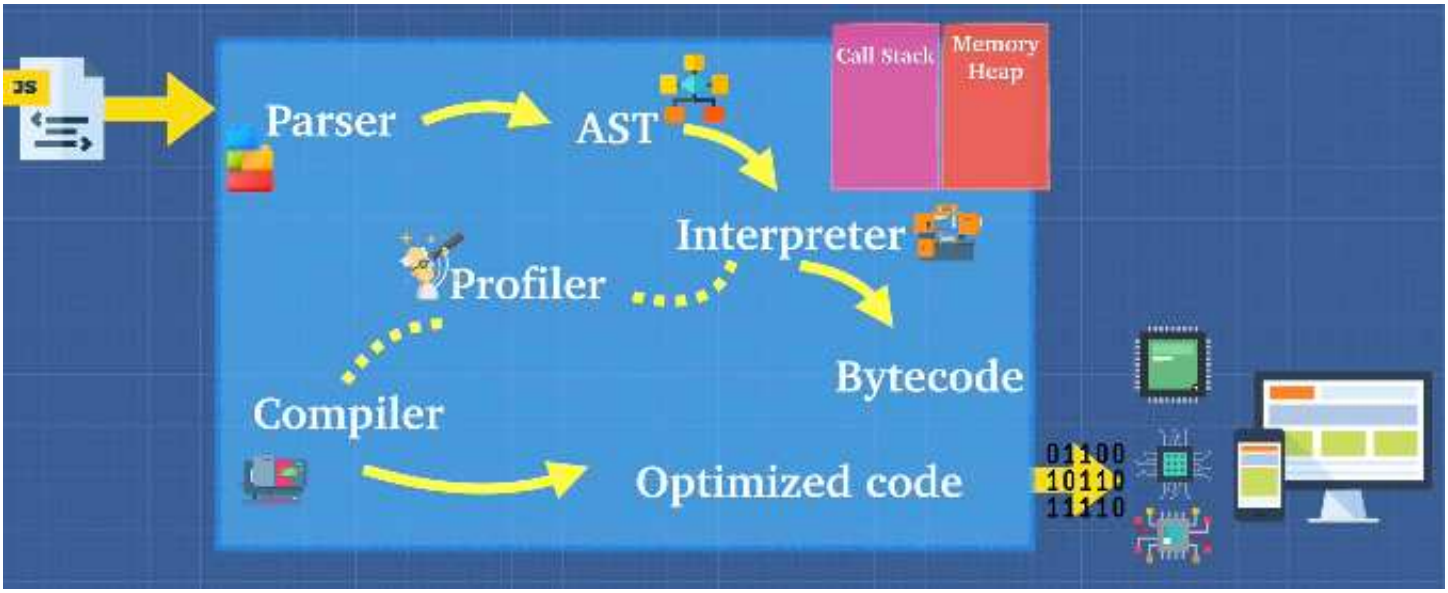


Kurs Xaritasi

JAVASCRIPT DASTURLASH MO'LINLARI

JavaScript dasturini berayotgan kompyuter dasturidir va unga JavaScript kodi beriladi va uni qanday bajarishini kompyuterka aytadi. Boshqacha aytganda, JavaScript va kompyuter tomonidan tushunadigan bir til orasidagi tarjimon. Lekin dastur ichida nima yuz beradi? Ha, bu dasturga qarab o'zgaradi. Bir nechta JavaScript Dasturlash Mo'linlari mavjud va odatda ular brauzer sotuvchilari tomonidan yaratiladi. Barcha dasturlash mo'linlari ECMA Script yoki **ES** tomonidan standartlashtirilgan.

JavaScript Dasturlash Mo'linlari Ro'yxati



> *Qisqa Snippet: 2008 yil Google Chrome V8 Dasturlash Mo'linini yaratganida JavaScript uchun ahamiyatli paydo bo'ldi. V8 mo'lini C++ tilida yozilgan ochiq manba'li yuqori samarali JavaScript mo'lini hisoblanadi va uni Chrome brauzeri va Node JS kuchini ta'minlaydi. Uning samarasi avvalgi barcha mo'linlarni o'tkazdi, asosan, uni tarkibida tarjimon va kompilyator mavjud bo'lganligi sababli. Bugun, barcha asosiy mo'linlar bu o'zgartirishni qo'llaydi.*

PARSER

Parsing kodni tahlil qilish, xatolarni tekshirish va uning qismlarga bo'lish jarayonidir.

AST

Parser **Abstract Syntax Tree** yoki **AST** deyilgan ma'lumotlar strukturasi yaratadi. AST asl sintaksisning har bir tafsilotini ko'rsatmaydigan, lekin tuzilish yoki tarkibga oid tafsilotlarni o'z ichiga olgan kodning daraxti grafidir. Ağaçda ba'zi narsalar zimmasiz va ko'rsatilmaydi, shuning uchun nomi so'zda abstract.

INTERPRETER

Interpretator har bir qatorni qator bo'ylab to'g'ridan-to'g'ri bajaradi, ularni mashina tiliga dastur sifatida kompilyatsiya qilishni talab qilmaydi. Interpretatorlar ishni sifatini oshirish uchun turli usullardan foydalanishlari mumkin. Ular manba koddan o'qib, uning o'zaro ishlab chiqilgan mashina kodi sifatida tarjima qilishlari, kompilyator tomonidan yaratilgan oldindan kompilyatsiya qilingan kodni bajarishlari yoki ulardan tashkil topgan kombinatsiyalardan foydalanishlari mumkin. V8 engine-da interpretator bayt-kodni chiqaradi.

> ***Ni#y Snippet:*** *Brendan Eich tomonidan 1995-yilda JavaScript yaratuvchisi sifatida Netscape navigator veb-brauzeri uchun yozilgan birinchi JavaScript engine.*
> *Boshlang'ichda, JavaScript engine faqat turlash interpreterdan iborat edi. Keyinroq bu SpiderMonkey engine ga aylanib, haliyam ham Firefox brauzeri tomonidan ishlatilmoqda.*

KOMPILEYATOR

Kompilyator kompyuter tomonidan o'qilishi va bajarilishi mumkin bo'lgan instruksiyalarni ma'mul-kod yoki past darajadagi shaklga aylantirish uchun oldin ishlaydi. Ushbu kompilyator barcha kodni ishga tushiradi va kodning nima qilishini aniqlab, uni kompyuter uchun o'qish qulay bo'lgan boshqa bir tilga aylantiradi. Babel yoki TypeScript haqida eshitgansizmi?
Ular JavaScript ekosistemasida keng qo'llaniladi va sizga yaxshi tasavvur bormi

ular haqida. Babel sizning zamonaviy JS kodlaringizni olishi va brauzer uchun mos JS (qariyb JS kod) qaytaradi. Typescript JavaScriptning ustunlikdagi turi bo'lib, JavaScriptga aylantiriladi. Ular ikkalasi ham to'g'ridan-to'g'ri kompilyatorlar qiladigan ishni bajaradi. Bir tilni oladi va boshqa bir tilga aylantiradi!

KOMBINATSIYA

Hozirgi engine-larda, interpretator kodni qator-qator o'qiydi, profiler tezkor foydalaniladigan kodni kuzatadi va uni kompilyatorga optimallashtirish uchun o'tkazadi. Oxirida, JavaScript engine interpretator chiqaradigan bytecode ni oladi va kompilyator chiqaradigan optimallashtirilgan kodni uni bilan aralashtiradi va undan so'ng uni kompyuterga beradi. Bu "Just in Time" yoki JIT kompilyatori deyiladi.

> ***Nifty Snippet:*** *Back in 1995 we had no standard between the browsers for compiling JavaScript. Compiling code on the browser or even ahead of time was not feasible because all the browsers were competing against each other and could not agree on an executable format. Even now, different browsers have different approaches on doing things. Enter WebAssembly a standard for binary instruction (executable) format. Keep your eye on WebAssembly to help standardize browsers abilities to execute JavaScript in the future!* *WebAssemby*

WRITING OPTIMIZED CODE

We want to write code that helps the compiler make its opBmizaBons, not work against it making the engine slower.

MemoizaAon

MemoizaBon is a way to cache a return value of a funcBon based on its parameters. This makes the funcBon that takes a long Bme run much faster aper one execuBon. If the parameter changes, it will sBll have to reevaluate the funcBon.

```
// Bad Way
function addTo80(n) {
  console.log('long time...')
  return n + 80
}

addTo80(5)
addTo80(5)
addTo80(5)

// long time... 85
// long time... 85
// long time... 85

// Memoized Way
functions memoizedAddTo80() {
  let cache = {}
  return function(n) { // closure to access cache obj
    if (n in cache) {
      return cache[n]
    } else {
      console.log('long time...')
      cache[n] = n + 80
      return cache[n]
    }
  }
}
const memoized = memoizedAddTo80()

console.log('1.', memoized(5))
console.log('2.', memoized(5))
console.log('3.', memoized(5))
console.log('4.', memoized(10))

// long time...
// 1. 85
// 2. 85
// 3. 85
// long time...
// 4. 90
```

Here are a few things you should avoid when wriBng your code if possible:

- eval()
- arguments
- for in

- with
- delete

There are a few main reasons these should be avoided.

JavaScript Hidden Classes and Inline Caching in V8

Managing Arguments

Inline Caching

```
function findUser(user) {
  return `found ${user.firstName} ${user.lastName}`
}

const userData = {
  firstName: 'Brittney',
  lastName: 'Postma'
}

findUser(userData)

// if this findUser(userData) is called multiple times,
// then it will be optimized (inline cached) to just be 'found Brittney Postma'
```

If this code gets opBmized to return only 1 name, then the computer would have to do a lot more work if you needed to return a different user.

Hidden Classes

```
function Animal(x, y) {
  this.x = x;
  this.y = y;
}

const obj1 = new Animal(1, 2);
const obj2 = new Animal(3, 4);

obj1.a = 30;
obj1.b = 100;
obj2.b = 30;
obj2.a = 100;

delete obj1.x = 30;
```

By ser ng these values in a different order than they were instanBated, we are making the compiler slower because of hidden classes. Hidden classes are what the compiler uses under the hood to say that these 2 objects have the same properBes. If values are introduced in a different order than it was set up in, the compiler can get confused and think they don't have a shared hidden class, they are 2 different things, and will slow down the computaBon. Also, the reason the delete keyword shouldn't be used is because it would change the hidden class.

```
// This is the more optimized version of the code.

function Animal(x, y) {
  // instantiating a and b in the constructor
  this.a = x;
  this.b = y;
}

const obj1 = new Animal(1, 2);
const obj2 = new Animal(3, 4);

// and setting the values in order
obj1.a = 30;
obj1.b = 100;
obj2.a = 30;
obj2.b = 100;
```

Managing Arguments

There are many ways using arguments that can cause a funcBon to be unopBmizable. Be very careful when using arguments and remember:

Safe Ways to Use Arguments

- arguments.length
- arguments[i] when i is a valid integer
- NEVER use arguments directly without .length or [i]
- STRICTLY fn.apply(y, arguments) is ok

CALL STACK AND MEMORY HEAP

The JavaScript engine does a lot of work for us, but 2 of the biggest jobs are reading and execuBng it. We need a place to store and write our data and a place to keep track line by line of what's execuBng. That's where the **call stack** and the **memory heap** come in.

Memory Heap

The memory heap is a place to store and write informaBon so that we can use our memory appropriately. It is a place to allocate, use, and remove memory as needed. Think of it as a storage room of boxes that are unordered.

```
// tell the memory heap to allocate memory for a number
const number = 11;
// allocate memory for a string
const string = "some text";
// allocate memory for an object and it's values
const person = {
  first: "Brittney",
  last: "Postma"
};
```