Hogyan sajátíthatom el a JavaScript nyelvet a leghatékonyabban?

Hello, itt Zsolt. Üdvözöllek az első fejezetben. Ezt a fejezetet azért írtam, hogy a helyes útra tereljelek vele és nagyban megkönnyítsem vele a JavaScript elsajátítását.

Rengeteg szoftverfejlesztővel és tanulóval foglalkoztam, és észrevettem, hogy sokan nagyon kemények vagytok önmagatokkal. Igazságtalannak tartom, hogy értékes emberek napról napra keményen ostorozzák magukat, és olyan tanulási stratégiát választanak, amely garantálja, hogy lassabban haladnak a kelleténél. Ha ezt a fejezetet olvasod, esélyed van arra, hogy veled ez ne forduljon elő.

Tartós javuláshoz a tanulás, az információ önmagában nem elég. Sőt, a túl sok információ egyenesen hátráltat. Gyakorlatra van szükség.

Miért más ez a könyv, mint a többi?

Rengeteg könyv pusztán elméleti megközelítésben tárgyal egy témát. Mit érsz vele? Talán az anyag pár százalékára emlékszel, néha egy-két téma beugrik, amikor problémákat oldasz meg.

Más könyvek inkább gyakorlati beállítottságúak, ahol az olvasó a szerző utasításai alapján épít egy vagy több programot. Napjainkban ezekre az anyagokra egyértelműen nagyobbnak látszik az igény, mint az elméletre, mert a Google keresőmotor az információt egyre könnyebben elérhetővé teszi.

Ma már nem az információ az érték, hanem a transzformáció. Az én célom is az, hogy pozitív változást érjek el a JavaScript tudásodban.

Bár találtam néhány kiváló gyakorlati könyvet, amelyek rám is transzformáló hatással voltak, sokszor hiányoltam ezekből a könyvekből, hogy megértésükhöz gyakran további kutatásra volt szükségem, amely nem a tudásomat gyarapította, hanem csak nehézségek elé állított. A legtöbb igazi gyakorlati könyv nem kezdőknek szól. Amikor az ES6 in Practice könyvet írtam, én is erre jöttem rá. Sok jó visszajelzést kaptam, de a kezdő szoftverfejlesztők nehézségekbe ütköztek. Ezek a visszajelzések indítottak el engem azon az úton, hogy megtaláljam, mi segít leginkább a kezdő szoftverfejlesztőknek.

Többféle anyaggal találkozol a könyvben:

1. **Ismeretanyag**: ezt többszöri olvasással és jegyzeteléssel tudod elsajátítani.
2. **Példák**: ezek az ismeretanyag részét képezik. A feladatod, hogy megértsd és kövesd őket. Első olvasatra nem szükséges önállóan megoldanod őket, elég, ha begépeled és futtatod a kódot, valamint megérted, hogy mi miért történik. Második olvasatra már a példákat is önállóan oldhatod meg.

3. **Gyakorlatok**: ezek a feladatok rád várnak. Oldd meg őket mielőtt megnéznéd a megoldást. Ha elakadsz, egy ötletért elkezdheted olvasni a megoldást, de amint eszedbe jut valami, próbálkozz tovább.
4. **Ellenőrző kérdések**: ezek a feleletválasztós kérdések ellenőrzik, hogy jó úton haladsz-e. A jegyzeteidet nem használhatod, csak a kérdést és a válaszokat olvashatod el. Ha hibáztál, menj vissza az adott fejezet ismeretanyagához és nézd meg, hogy mit nem értettél. Készíts róla jegyzetet.

Hogyan olvasd ezt a könyvet?

A könyv egyszeri elolvasása, kipipálása, majd egy újabb könyv vagy egyéb oktatóanyag keresése az egyik legkevésbé hatékony stratégia, amelyhez folyamodhatsz.

Tudom, sok tanulnivalód van, főleg ha most kezded a szoftverfejlesztést. Viszont pont ezért érdemes extra energiát fektetned az ismeretek mélyebb elsajátításába.

Az alábbiakban kérdés-válasz formátumban fejtem ki, hogy hogyan hozhatod ki a maximumot ebből a könyvből.

Kérdés: Minek a jegyzetelés?

Válasz: Az agyunk gyorsabban áll rá a témára, több kapcsolatot hoz létre az idegsejtjeink között az adott témában.

Kérdés: Hogyan érdemes jegyzetelni?

Válasz: Egy jó tollal egy olyan füzetbe, amelybe öröm írni. Ne sajnáld rá a pénzt. Később magaddal viheted és átismétled. Az agyad gyakran emlékszik arra, hogy mit gondoltál éppen akkor, amikor egy adott oldalt írtál. Csak ránézel az oldalon lévő szöveg elrendezésére, és beugrik, hogy akkor mi járt a fejedben. Ezt leginkább a tollal való jegyzetelés hozza ki belőled.

Kérdés: Szövegszerkesztőbe is jegyzetelhetek?

Válasz: Igen. Valószínű okkal kérdezed és tisztában vagy vele, hogy a fenti előnyökről lemondasz akkor, amikor nem fizikai jegyzetet készítesz, hanem egy szövegszerkesztőben jegyzetelsz. Sokkal jobb szövegszerkesztőbe írnod, mintha egyáltalán nem jegyzetelnél. Emellett bármikor bárhová magaddal viheted a jegyzeted, ha van internetelérésed.

Kérdés: Mit jegyzeteljek?

Válasz: Általában egy téma vázát érdemes leírnod. Az ismert kulcsszavakról eszedbe fog jutni, hogy mit tanultál. Ha egy alapfogalommal nem vagy tisztában, írd le a meghatározását. Ha valami eszedbe jut, írd le. Ha valamit nem értesz, nézz utána, és írd le.

Kérdés: Hogyan ellenőrizhetem a tudásom?

Válasz: Gyakorlatokkal és ellenőrző kérdésekkel (ld. fent). Ha valamit elrontasz, semmi gond. Ismételd át a témát.

Kérdés: Egy "guru" azt tanította nekem, hogy semmi értelme egy könyvet az elejétől a végéig elolvasni. Miért ajánlod nekem mégis, hogy végigolvassam a könyved?

Válasz: Azért vetted ezt a könyvet, mert el akarod sajátítani a JavaScript alapjait? Ez esetben az alapok lefektetése akkor jó, ha később stabilan támaszkodhatsz rájuk. Ha középhaladó vagy haladó vagy, van számodra egy másik könyvem. Ha ebben a haladóbb könyvben megakadsz, javaslom, hogy térj vissza az alapokhoz és kövesd a programot.

Többen tanítják azt, hogy egy könyvet nem érdemes elejétől a végéig olvasni, hanem lapozgatni kell őket és a releváns részeket tanulmányozni. Bár ez a megközelítés gyakran gyors győzelmeket eredményez, egy alapozó könyvnél ezek a gyors győzelmek nem vezetnek célra. Itt a cél az, hogy a JavaScript alapjait elsajátítsd. Nem az, hogy még ma tudj készíteni egy webalkalmazást. Erre is készül könyvem, de előfeltétele a JavaScript ismerete.

Kérdés: A gyakorlatokat mindenképp meg kell oldanom?

Válasz: Ha el szeretnéd sajátítani a JavaScript alapjait, egyértelműen igen. Eleinte lehet, hogy nehéz lesz rávenned magad. Beismerem, sokszor ez olyan, mintha fizikai fájdalmat éreznél. De idővel rájössz, hogy a JavaScript kódolás olyan, mintha játszanál. Ahhoz, hogy ezt az áttörést elérd, csinálnod kell a feladatokat. Ellenkező esetben a haladásod megkérdőjelezhető.

Időközönkénti Ismétlés

Az Időközönkénti Ismétlés (Spaced Repetition) technika lényege, hogy egyre ritkábban idézed fel a tanulandó ismeretanyagot. Ezáltal gyakran éppen azelőtt találkozol az ismerettel, hogy elfelejtenéd.

A jegyzetedből készíthetsz kártyákat, amelyeket időről időre átnézhetsz. Ha véletlenszerű sorrendben nézed át a kártyáidat, az agyad nem fogja mankóként használni az egymáshoz nem tartozó témák sorrendjét. Ezáltal magára az ismeretre fogsz emlékezni és nem arra, hogy én hogyan álmodtam meg a fejezetek sorrendjét.

Mivel fizikai kártyákat készíteni időigényes, javaslom az Anki alkalmazás használatát. Az Anki segítségével könnyebben tudsz új ismeretekre szert tenni és segít arra, hogy hosszútávon is emlékezz rájuk.

Tanulási Terv Kezdő Szoftverfejlesztőknek

A könyvet legalább háromszor érdemes végigolvasnod.

Első olvasáskor végigmész az ismeretanyagokon, jegyzetelsz, figyelsz a példákra. Nem oldod meg a példákat önállóan, de a könyvvel együtt begépeled őket. Próbálj meg kreatívan egy-két dolgot változtatni a kódban, és nézd meg, hogy mi történik. A gyakorlatokat önállóan oldd meg. Ha elakadsz, keress egy ötletet a megoldásban, de amint hozzájutottál egy használható ötlethez, fejezd be a megoldás olvasását és próbálkozz tovább. Amint kész vagy egy gyakorlattal, fusd át a megoldást, és jegyzetelj, ha új dolgot

tanultál. Az ellenőrző kérdésekkel ellenőrizheted a tudásod, ha rontasz, jegyzetelj, és készíts a hibás válaszod alapjául szolgáló ismeretanyaghoz egy Anki kártyát.

Második olvasáskor már a példákat is önállóan oldd meg. Ha egy gyakorlatban elakadsz, a megoldás helyett nézd meg az elméleti részt. Ha nem megy így se a megoldás, keress Google-n. Kövess el mindent azért, hogy meg tudd oldani a feladatot a megoldás megnézése nélkül. Emlékezz, egyszer már láttad a megoldást. Minél erősebben próbálkozol, annál mélyebb lesz az emlék.

A harmadik olvasás jellemzően sok idővel, akár fél évvel később következik be. Csodálkozni fogsz, hogy mennyi emléked lesz a könyvről. Itt már az egyértelmű részeket és egyértelmű feladatokat nyugodtan átugorhatod. A lényeg, hogy minden gyakorlatot és ellenőrző kérdést önállóan tudj megoldani.

Tanulási Terv, Ha ez a Sokadik Programnyelved

Rengeteg fogalom máshonnan ismerős lesz számodra. Ezért a problémamegoldó képességed fejlesztése és a JavaScript alapjainak az elsajátítása lesz neked fontos.

A tanulási terved hasonló ahhoz az esethez, mintha ez lenne az első programnyelved, csak itt semmi szükség olyan témákat jegyzetelned, amelyeket más programnyelvekkel kapcsolatban már megtanultál.

Bevezetés a JavaScript Nyelvbe

Ezt a könyvet azért írtam, mert nem könnyű egy új programozási nyelvet megtanulni. Ez kiváltképp igaz akkor, ha ez a nyelv az első programozási nyelvünk. Szándékosan olyan nyelvezetet használtam, hogy semmilyen JavaScript előképzettségre ne legyen szükséged ahhoz, hogy megértsd a nyelvet.

Mielőtt belevágnánk a tanulásba, nézzük meg, hogy miért is érdemes a JavaScript nyelvvel foglalkozni napjainkban. Ebből a bevezető fejezetből megtudhatod, hogy

- miért ma a legjobb elkezdeni a JavaScript nyelvvel foglalkozni,
- milyen karrierlehetőségeid vannak,
- miért könnyebb napjainkban a JavaScript nyelvet elsajátítani, mint a múltban bármikor.

JavaScript a Böngészőben

Valószínű a weboldalamra egy web-böngészőn keresztül találtál, mint pl. a Google Chrome vagy a Firefox. A web-böngészőben megjeleníthető weboldalakat három nyelven írják: HTML, CSS, és JavaScript.

A HTML (Hyper Text Markup Language) nyelv a weboldal szerkezetét és tartalmát írja le. A CSS (Cascading Style Sheets) a megjelenítéssel foglalkozik: meghatározza, hogy milyen betűtípussal jelenjen meg a szöveg, milyen színek jelenjenek meg az oldalon, háttérképeket illeszt be megfelelő helyekre.

A JavaScript pedig egy programozási nyelv, amely a weboldalon lévő elemek mozgatásán, változtatásán kívül rengeteg más feladatot láthat el. Néhány példa:

- kommunikáció egy szerverrel, amely az adataidat tárolja,

- alkalmazás-logika leírása: például képes arra, hogy ellenőrizze, hogy helyesen írtad-e be a személyes adataidat,
- vizualizáció: például kirajzolni az OTP részvény árfolyamát.

> Az *animáció* a képernyőn lévő elemek mozgatása és transzformálása. A mozgatás alatt az objektum jellemzői változhatnak. Az objektum színe, mérete változhat, sőt három dimenziós objektum animálásakor az objektumot elképzelhető, hogy idővel egy másik szögből látjuk.

Transzformálás alatt a vizsgált elem valamilyen jellemzőjének időbeli változtatását értjük. Megváltozhat egy elem színe, formája, helyzete.

Az előbb megemlítettük a szervert mint fogalmat. Itt az idő meghatározni, hogy mit értünk kliens oldali programozás és szerver oldali programozás alatt:

> A *kliens oldali programozás* a felhasználó böngészőjében futó kóddal foglalkozik. A *szerver oldali programozás* a háttérben futó kóddal foglalkozik, amely szolgáltatásokat biztosít a kliens számára.

Amikor lekérjük az OTP részvény adatait egy szervertől, egy szerver oldali program lekérdezi az adatokat egy adatbázisból. Ezeket az adatokat a kliens egy meghatározott formátumban (pl. XML, JSON) kapja meg. Miután a kliens elolvasta az adatokat, ki tud rajzolni egy grafikont a böngészőbe.

A JavaScript egyértelműen a kliens oldali programozás egyszerű nyelve volt a kilencvenes években és a kétezres évek elején. A JavaScript kizárólagos felhasználási területe a weboldalakon lévő elemek animálása volt. A JavaScript célja *dinamikus tartalom* létrehozása volt.

Az animációhoz kapcsolódik a *renderelés* fogalma.

> Amikor egy adott időpillanatban megjelenítünk egy objektumot a képernyőn, az objektumot *rendereljük*.

Egy objektum animációja nem más, mint egy objektum folyamatos időbeli renderelése, miközben az objektum jellemzői változhatnak.

Húsz évvel ezelőtt a szoftverfejlesztők egyáltalán nem vették komolyan a JavaScript nyelvet. Ez azoknak a kontároknak volt köszönhető, akik mindenféle tudás nélkül csak össze-vissza másolgattak olyan JavaScript kódokat, amelyekhez lövésük sem volt, hogy mit jelent. Az Interneten kerestek egy animációt, és bemásolták a JavaScript kódját. Mivel két animáció kódja könnyen kölcsönhatásba léphetett egymással, sokszor működőképes animációkat tett tönkre egy frissítés.

A JavaScript népszerűsége az *AJAX* (Asynchronous JavaScript And XML) elterjedésével nőtt. A 2006-os év hozta meg az `XMLHttpRequest` objektum szabványosítását, amely lehetővé tette, hogy az oldal újra betöltése nélkül kérj adatokat egy szervertől. Korábban minden szerver-kommunikáció weboldalak újratöltésével járt, és ez az alacsony sebességű modemek idejében több másodpercbe is telhetett. Így az AJAX népszerűsége érthető: a felhasználók úgy érzékelték, hogy azonnal hozzájuthatnak az adataikhoz.

Ahogy a JavaScript nyelv egyre inkább elterjedt, a JavaScript programozó közösség újabb problémával kellett, hogy szembenézzen: ahhoz, hogy minden böngészőben futó kódot tudjunk írni, különböző böngészőkhöz más-más JavaScript megoldásokat kellett alkalmazni. Néha a helyzet kétségbeejtően zavart volt. Ahhoz, hogy minden fontosabb böngészőben jól működő kódot tudj írni, JavaScript-szakértővé kellett válnod.

Idővel egy másik újítás ezt a problémát is megoldotta. Szintén 2006-ban a JavaScript történetének újabb mérföldköve jött el: megjelent a `jQuery` könyvtár. A jQuery szlogenje *write less, do more*. Magyarul szabad fordításban *írj kevesebbet, érj el többet*. Sokszor tíz-húsz soros böngészőről böngészőre optimalizált kódot egyetlen rövid jQuery paranccsal helyettesíthettünk. Ennek következtében már nem volt szükség arra, hogy komoly szoftverfejlesztői tapasztalattal rendelkezz ahhoz, hogy működőképes JavaScript kódot tudj írni. Elég volt a jQuery könyvtárat használnod, és a legtöbb problémád meg is oldódott.

Sajnos a jQuery egy új problémát hozott létre: túl könnyű lett valós programozási ismeretek nélkül jQuery kódot írni, így azok is kódolni kezdtek, akik nem igazán tudták átgondolni, hogy mit is csinálnak.

Mindeközben a *single page application (SPA)* és a *client-side rich web-application* kifejezések születtek meg, amelyek a kliensoldalon egyszer betöltött webalkalmazásokat jelölték, amelyek oldalfrissítés nélkül, egyetlen oldal betöltése után működtek. Mi az *újratöltés nélküli webalkalmazás* kifejezést fogjuk használni a jelenség leírására.

Az AJAX és a jQuery előtt szinte mindent szerver oldali kód végzett. A szerver egy weboldal kódját állította elő, és ezt küldte el a kliensnek HTML, CSS, és JavaScript kód formájában. A kliensnek mindössze annyi feladata volt, hogy betöltse ezt a kódot. Az újratöltés nélküli webalkalmazások elterjedésével a weboldal renderelése és az alkalmazás logikájának egy része a kliens oldalra került. A JavaScript kódot az új módszer alapján egyszer kell betölteni. Ez gyakran több megabyte méretű alkalmazáskódot jelentett. Ekkorra már a kliens oldalon elég erős számítógépek voltak ahhoz, hogy megbirkózzanak a kliens oldali JavaScript futtatásával.

> Az *újratöltés nélküli webalkalmazások* az oldal betöltésekor minden HTML, CSS, és JavaScript kódot megkapnak, és anélkül is tudnak navigálni a különböző oldalak között, hogy erről a szervert informálnák. A kliens AJAX segítségével kommunikál a szerverrel, és úgy fér hozzá a szerver által biztosított adatokhoz, hogy nem kell újratöltenie a weboldalt. Így az újratöltés nélküli webalkalmazások gyorsabbak és könnyebben használhatóbbak lettek, mint a szerver oldalon megírt alkalmazások. Pont olyan könnyű lett az futtatásuk, mint egy Windowson, Linuxon, vagy Mac-en futó alkalmazásé.

A jQuery jelentősen leegyszerűsítette a webfejlesztést, és ezáltal sokkal többen próbálkoztak meg munkát és szabadúszó feladatokat elvállalni előképzettség nélkül. Kis feladatokra ez a megközelítés működőképes is volt. A felhasználói kör azonban hamar rájött, hogy a jQuery használata önmagában egy bizonyos méretű alkalmazás felett nem ajánlott, mert a szoftver karbantartása lehetetlenné válik.

> A *karbantarthatóság (maintainability)* egy szoftver könnyű fejleszthetőségét, módosíthatóságát, skálázhatóságát, tesztelhetőségét, és a

> hibák könnyű eltávolítását (debuggolhatóság) jelöli. Ha tíz sor kódot írsz, más módszerekre lesz szükséged, mint egy millió soros program írásakor.

A fenti ötösből a skálázhatóság azt jelenti, hogy megnövekedett terhelésre a program kiszámítható választ ad. A skálázhatóságot érthetjük erőforrásra és fejlesztési módszerekre is. Például ha egy szerver száz felhasználóra 1 másodperc átlagos válaszidőt garantál, akkor szeretnénk tudni, hogy mennyi idő alatt kapunk választ, ha egyszerre ezer, tízezer, vagy százezer felhasználó csatlakozik a szerverhez. Egy fejlesztési folyamat skálázhatóságánál nézhetjük például azt, hogy hogyan növekszik egy hiba kijavításának az ideje tízezer, százezer, vagy millió soros programnál, illetve hogy ezer soronként hány hibát vétenek a fejlesztők.

A karbantarthatósági problémára a válasz *JavaScript keretrendszerek (JavaScript frameworks)* formájában érkezett. A keretrendszerek úgy strukturáljak a kódot, hogy az karbantartható legyen, mindezt a szoftvermérnökök által is támogatott formában. Ahhoz, hogy a keretrendszereket megfelelően alkalmazni tudjuk, szoftverfejlesztői tudásra van szükségünk. A szabadúszó portálokon ez a tudás többet ér, mint a jQuery.

> Egy *keretrendszer* (pl. Angular, React) a kódodat strukturálja. Használatával átadjuk az irányítást a keretrendszernek. Ez azt jelenti, hogy a keretrendszer hivatkozik a kódodra.

> Egy *könyvtár* (pl. jQuery) olyan funkciók összessége, amelyeket a programunk használ feladatok megoldására. A könyvtár nem keretrendszer, mert az irányítás nálad marad. Egy könyvtár nem hivatkozik az általad írt kódra.

A 2010-es évek elején szinte mindenki JavaScript keretrendszert akart írni. Hetente jöttek ki az újabbnál újabb keretrendszerek. Ezáltal sokan újra feltalálták a kereket azáltal, hogy egy jól ismert problémát egy újabb módon oldottak meg. Ha most egy szoftverfejlesztői portfóliót szeretnél építeni, nem javaslom, hogy keretrendszert írj.

Szerencsére az újabbnál újabb keretrendszerek gyors népszerűsége és még gyorsabb bukása már a múlté. Ma már nem kell újabbnál újabb keretrendszereket és könyvtárakat tanulnunk havi rendszerességgel. Napjainkban népszerű a React, az Angular, és a Vue. A Google keresési adatok alapján a React áll nyerésre, de az Angular-nek és a Vue-nek is megvan a maga felhasználási területe. Mindhárom keretrendszer mögött hatalmas nyílt forráskódú közösség áll. A React mögött a Facebook, az Angular mögött a Google áll.

> *Nyílt forráskódú közösségekhez* bárki csatlakozhat, hiszen a a közösség nyílt. Bárki hozzájárulhat a közösség által karbantartott szoftver forráskódjához. Ez adja a megfelelően népszerű nyílt forráskódú közösségek erejét. A főbb hibákat gyorsan kijavítják, és maga a szoftver folyamatosan fejlődik. Ezáltal népszerű nyílt forráskódú közösségek iránt tanúsíthatunk egyfajta bizalmat, ami egy adott minőségi szintet garantál.

Bár a keretrendszerek elterjedése kétségkívül hasznos lépés volt, a szoftverfejlesztők még egy problémával néznek szembe napjainkban. A keretrendszerek és könyvtárak olyan mértékben gazdagították a JavaScript nyelvet, hogy a JavaScript nyelv részletes elsajátítására már nem is volt szükség. A JavaScript ES5 verziója 2010-es évek közepére már egyébként is idejétmúlt volt, és szükségesnek látszott egy nagyobb

ráncfelvarrás. Ezért kijött az ES6 vagy más néven ES2015 változat. Ez az újabb változatot használva rengeteg különböző forrásból táplálkozott: több könyvtár, mint pl. a jQuery és az Underscore, több keretrendszer, több nyelv, mint pl. a TypeScript újdonságait tartalmazta, és lényegesen megkönnyítette a JavaScript nyelv használatát. Kevesebb kóddal sokkal többet lehetett kifejezni.

> Az ES az ECMAScript rövidítése. AZ *ECMA International* (European Computer Manufacturers Association) egy 1961-ben alapult szabványosító szervezet, amely többek között a JavaScript és az ActionScript nyelvek szabványosítása mögött áll. A JavaScript szabványt az ECMA-262 írja le.

Egyben a régi böngészők, mint pl. az Internet Explorer 8, kimentek a divatból, és ezáltal néhány JavaScript megoldásra nem volt már szükség. Ezáltal több kompatibilitási megoldás szükségtelenné vált, és a JavaScript kódunk egységesebbé és tömörebbé vált. Az újabb böngészők egyre inkább ragaszkodtak a legújabb ECMAScript szabványhoz, ezáltal nem volt szükség különböző böngészőkben különböző megoldásokra.

Az ES2015 megjelenése óta évről évre fejlődik a JavaScript nyelv. Az angol nyelvű ES6 in Practice könyvemet és videotanfolyamomat így évről évre frissítettem. Talán mire erre a hivatkozásra kattintasz, már át is neveztem "ES6+ in Practice" vagy "JavaScript in Practice (2019 Edition)-re" a címet.

A JavaScript fejlődési története alapján levonhatjuk a következtetést, hogy még soha nem volt jobb döntés JavaScript fejlesztőnek állni, mint napjainkban. Sok munkahelyen több pénzt visz haza egy JavaScript fejlesztő, mint egy C++ vagy PHP fejlesztő társa. Ha nem hiszel ebben a mondatban, nézz utána payscale.com-on, vagy böngészd át a stackoverflow.com-on megjelenő álláshirdetéseket.

Egy JavaScript fejlesztő napról napra érdekes kihívásokat talál, és egy feladat megoldása közben azonnali visszajelzést kap arról, hogy működik-e a kódja vagy nem. Hiszen napjaink fejlesztőkörnyezeteiben már a böngészőt sem kell manuálisan frissítenünk, a *live reload* funkció élőben tölti újra a frissített kódot.

> Összefoglaló:
>
> - A JavaScript-et eleinte sokan komolytalan animációkat leíró nyelvnek tartották
> - Az AJAX segítségével kommunikálhattunk a szerverrel anélkül, hogy újratöltenénk a böngészőben futó oldalt
> - A jQuery elterjedése jelentősen megkönnyítette a webfejlesztést
> - Keretrendszerek és könyvtárak rákényszerítették a fejlesztőket arra, hogy könnyen karbantartható kódot írjanak
> - A régi böngészők elavulásával könnyebb lett JavaScript nyelven programot írni, mert az új böngészők egyre jobban ragaszkodtak a JavaScript szabványához.
> - Az ES2015 után évente frissült a JavaScript nyelv, és sok könyvtárakban, keretrendszerekben, és más nyelvekben használt megoldás beépült a JavaScript nyelvbe

JavaScript a szerver oldalon

Ha a fenti előnyök nem lennének elegek ahhoz, hogy JavaScript fejlesztést tanulj, ez a fejezet további bíztató információkkal szolgál.

A JavaScript nem csak a kliens oldalon használható. A node.js használatával szerver oldalon is futtathatunk JavaScript kódot. A node.js a Google Chrome alapjául szolgáló motor alapján épült. Ennek a motornak a neve V8. A V8 motor előnye, hogy gyors, mint egy Forma-1-es autó. Emlékezzünk vissza, Michael Schumacher is V8-as motorral nyerte meg az első világbajnoki címét. A V8 *fordító (compiler)* a JavaScript kódot gépi kóddá alakítja, ezáltal szerver oldalon már nem *interpretált* kódot, hanem gépi kódot futtathatunk.

> Az *interpretált kód* egy *virtuális gépen* fut. A virtuális gép a fejlesztő által írt kódot a számítógép által érthető utasításokká alakítja. Az interpretált nyelvek általában lassabbak, mint a *gépi kóddá fordított (compiled)* nyelvek. Az interpretált nyelvek másik előnye a *hordozhatóság*, ami azt jelenti, hogy az interpretálható forráskódot különböző környezetekben különböző virtuális gépek segítségével futtathatjuk. A gépi kóddá fordítható nyelvek előnye, hogy a lefordított kód gépközelibb, ezáltal jobban kihasználja az adott számítógép rendszerének lehetőségeit. Ezáltal a lefordított kód hordozhatósága el is veszik, mert különböző operációs nyelvek nem ismerik egymás belső felépítését.

A node.js azáltal lett gyors, hogy egy fordító gépi kóddá alakítja a JavaScript kódot. Ez önmagában még nem jelent sokat, mert sok más szerver oldali nyelv képes ugyanerre. Így feltehetjük a nyilvánvaló kérdést: miért éri meg node.js-ben fejleszteni szerver oldali alkalmazásokat?

A node.js népszerűségének oka az architekturájában rejlik. A node.js egy *eseményciklust (event loop)* használ a párhuzamos szerver oldali kérések (*request*) ütemezésére. Mivel a véleményem szerint az eseményciklus nem túl jó szakkifejezés, a továbbiakban event loop-ként fogunk hivatkozni rá. Fölösleges mindent magyarosítani, hiszen a túlzott magyarosításnak az lesz az eredménye, hogy elszigeteljük magunkat, és ezáltal drasztikusan csökkennek a lehetőségeink.

> Az *architektúra* szó töve az *architect*, ami tervezőmérnököt jelent. A szoftver tervezőmérnök terméke *szoftver architektúra*, amely egy szoftver működési elvét leíró terv. Így legkönnyebben működési elvként hivatkozhatunk rá.

Az event loop-ot egy végtelen körfolyamként lehet elképzelni, ahol a párhuzamos folyamatok egy időszeletet kapnak. Ezáltal több párhuzamos folyamatot is futtatni tudunk anélkül, hogy az egyes folyamatokat blokkolódnának. Ezt nevezzük *blokkolásmentes I/O-nak (non-blocking I/O = non-blocking Input/Output)*.

> Az *I/O műveletekből* az I/O Input/Output-ot jelent. Az input beviteli, az output kiviteli művelet. A beviteli művelet az, amikor adatot viszünk be egy rendszerbe. Ilyen például a billentyűzeten való gépelés, az egér, a scanner, a fényképezőgép, a letöltés, és a banki egyenlegünk lekérdezése. A kiviteli művelet ennek az ellentéte. Kiviteli művelet során egy rendszer adatot szolgáltat, pl. megjeleníti a begépelt szöveget, nyomtat, feltölt egy

> másik szerverre, vagy megadja a banki egyenlegünket mint a kérésünkre adott választ egy adott formátumban.

Sok más nyelvvel ellentétben egy adatbázis vagy fájl írása közben a node.js nem blokkol más folyamatokat amíg az I/O művelet véget nem ér, hiszen ezalatt más feladatokat is elvégezhetünk, amelyek más ügyfeleknek vagy folyamatoknak fontosak lehetnek.

Gondolkodjunk el egy pillanatra. Mikor is van blokkolásmentes I/O-ra szükség?

> A node.js fő előnye abban rejlik, hogy sok párhuzamos I/O intenzív műveletet tud kezelni. A node.js más alternatívákhoz képest nem különösen erősebb olyan feladatokban, ahol sok számítást kell végezni.

Például egy három dimenziós renderelő algoritmus nagyon számításigényes. Bár a V8 motor gyors gépi kódot eredményez, más, gépközelibb nyelvek hatékonyabban képesek ugyanezt a feladatot megoldani. A node.js akkor válik hatékonyabb alternatívává, ha sok alacsony számításigényű műveletet kell párhuzamosan kezelni.

Napjainkban a legnépszerűbb alkalmazások legfőbb jellemzője pont a sok alacsony számításigényű feladat. A felhasználók számítógépei már elég erősek ahhoz, hogy az alkalmazási logika megjelenítési részével megbirkózzanak, és táblázatokat, grafikonokat, és egyéb komponenseket jelenítsenek meg. Az alkalmazás szerver egy *API-n (Application Programming Interface, szándékosan nem fordítom le)* keresztül biztosít adatokat. Ehhez adatbázisok és fájl szerverek szolgáltatásait veszi igénybe. A kliens oldali alkalmazás elsősorban AJAX hívásokkal kommunikál a szerver által biztosított API-val.

> *API (Application Programming Interface)*: egy alkalmazás használati módját, kommunikációs protokollját írja le egy külső felhasználó szemszögéből. Egy alkalmazás szerver egy *szolgáltatásának (service)* API-ja *végpontokból (endpoint)* áll, amelyeket egy *API request* (kérés) segítségével lehet meghívni. Általában az API request egy webcímet és megfelelő adatokat (*payload*, magyarul szakmaiatlanul hangzó hasznos teher) tartalmaz. Az API válasza (*API response*) adott formátumban prezentált információ, ami lehet JSON (JavaScript Object Notation) vagy XML (eXtensible Markup Language). Az API request payload és az API response általában ugyanabban a formátumban íródik. Ez a formátum JavaScript alkalmazások esetén a legtöbbször JSON. Az alkalmazás szerver API-jának a felhasználója a kliens oldali alkalmazás, vagy más szerver oldali alkalmazás lehet.

Napjaink API alapú alkalmazásai ritkán számításigényesek. Ebben a környezetben a node.js előnyei könnyen érvényre jutnak.

A *microservice*-ek (mikroszolgáltatás) sok problémát hatékonyan megoldanak. Szerver-szerver kommunikáció segítségével egyszerűen csak lekérdezhetünk egy számításigényes műveletet egy *microservice*-től, és ezt a szolgáltatást nem kell feltétlenül JavaScript-ben megírni. Ha ez a téma érdekel téged, akkor olvasd el a mellékelt angol nyelvű cikkemet.

Általánosságban levonható az a következtetés, hogy a kliens oldali programozás megtanulása után könnyen megtanulható a szerver oldali programozás is. *Full stack fejlesztőnek* hívjuk azokat a szoftverfejlesztőket, akik mind a kliens oldali, mind a szerver oldali programozáshoz értenek. A full stack fejlesztővé válás egy kiváló út a karrieredben, amennyiben több felelősségre vágysz egy kisebb cégben. Ezekben a cégekben a felelősségvállalás gyors kiugrási lehetőséget biztosít.

> Összefoglalás:
>
> - A JavaScript nyelv gyors gépi kóddá fordítható a Google V8 motorja segítségével. Ez a gépi kód szerver oldalon is képes futni.
> - A node.js blokkolásmentes módon kezeli az I/O műveleteket. Ezáltal sok I/O műveletet képes párhuzamosan kezelni.
> - Napjainkban sok számításigényes alkalmazást kliens oldalon vagy egy adott feladatra specializált microservice segítségével oldanak meg. Ezáltal a kevésbé erőforrásigényes, de sok párhuzamos kérés kiszolgálását lehetővé tevő API programozásra a node.js kiválóan alkalmas.

A JavaScript egyéb felhasználási lehetőségei

A JavaScript fejlesztés élvezetes. Eredetileg a JavaScript fő felhasználási területe a kliens oldali fejlesztés volt. Ezt követte a szerver oldali programozás node.js segítségével. Ha viszont körbenézünk, hogy milyen egyéb feladatokat képes a JavaScript ellátni, könnyen meglepődhetünk:

- **JavaScript és a virtuális valóság.** A React 360 segítségével virtuális valóságélményt hozhatunk létre JavaScript-ben. A React 360 mellett további VR könyvtárak állnak a rendelkezésedre.
- **A JavaScript más programozási nyelvekre fordítható, és más programozási nyelvek JavaScript-re fordíthatók.** Például a **WebAssembly** több nyelvet képes JavaScript-re fordítani, ezáltal más programozási nyelvben írt kódot futtathatunk a böngészőnkben és használhatunk fel a saját JavaScript kódunkban.
- **JavaScript machine learning (gépi tanulás) és mesterséges intelligencia.** A Google gépi tanulási funkciókat biztosító népszerű könyvtára a TensorFlow, elérhető JavaScript-ben is. A TensorFlow segítségével neurális hálókat hozhatunk létre JavaScript-ben. Sok más gépi tanulást és mesterséges intelligenciát elősegítő könyvtár elérhető JavaScript-ben.
- **Szoftverfejlesztés mobil eszközökre.** A React Native lehetővé teszi, hogy mobil eszközökre fejlesszünk alkalmazásokat JavaScript-ben. Ez egy nagyon hatásos kombináció, mert több eszközön is újrafelhasználhatjuk a programkódunkat.
- **Blokklánc-fejlesztés.** A fejezet írásának pillanatában több, mint ezer blokklánc (blockchain) témájú fejlesztés áll rendelkezésre npm-ben (bizonyíték). Például a Crypto-js könnyen használható titkosító és visszafejtő algoritmusokat ad. A rendelkezésre álló könyvtárak segítségével könnyedén létrehozhatunk JavaScript alapú blokkláncot.
- **Valós idejű kommunikáció a weben.** Mind videó, mind audio streaming szolgáltatások hozhatók létre JavaScriptben a WebRTC nyílt forráskódú projekt segítségével.
- **Microservice alkalmazások.** A Seneca egy lehetséges példa, amely microservice alapú alkalmazások létrehozását segíti.

A JavaScript bizonyos feladatok elvégzésére optimális, míg más feladatokra csupán alkalmas. Értelemszerűen a nagy számításigényű feladatok (pl. karakter animáció, 3D-s árnyékolás) elvégzésére a JavaScript messze nem optimális. Ugyanakkor a JavaScript képességei és futtatókörnyezete folyamatosan fejlődik, így újabbnál újabb felhasználási lehetőségek merülnek fel, ahol a JavaScript megbízható alternatívát nyújt más nyelvekhez képest. Amint egy felhasználási lehetőség megoldhatóvá válik JavaScript segítségével, számíthatsz egy méretes és lelkes nyílt forráskódú közösség támogatására, és akár ipari támogatottságra is.

JavaScript dominancia a munkaerőpiacon

Megjegyzés a magyar kiadáshoz: *Ennek a fejezetnek az első változata angol nyelven jelent meg, így itt most nemzetközi trendekkel foglalkozunk. Magyarország számos területen nagyjából öt évvel van elmaradva a nemzetközi trendekhez képest. Így a Python és a JavaScript helyett Magyarországon ma még a Java dominál.*

Ugyanakkor nem véletlenül teszem elérhetővé a magyar nyelvű anyagaim angol változatát. Hiszek abban, hogy a nemzetközi piacon több lehetőséged van megfelelő fizetés vagy szabadúszó munkák megszerzésére.

Mit is jelent ez a szempontodból? Mire igazán befutsz a karrieredben, feltehetőleg akkorra a magyar trendek elérik a mai nemzetközi helyzetet. Ezáltal az alábbi leírás kicsit a jövőt is előrevetítheti. Emellett ne feledd, hogy a JavaScript jelenleg egyeduralkodó a kliens oldalon, és a felhasználói élmény (UX - User Experience) előretörésével a JavaScript fejlesztés megkerülhetetlen az életben maradást megcélzó cégeknél. Remélem egy nap az Oracle is rájön erre, mert az előző munkahelyemen a HR rendszerük kezelése képzett szoftverfejlesztőknek is hatalmas fejtörést okozott.

A 2019-es HackerRank közvéleménykutatás szerint:

- Hatalmas a szakadék a munkaadói kereslet és a fejlesztők keretrendszerekben való jártassága között. A legnagyobb szakadék Reactban volt: a munkaadók 38,38%-a keresett React fejlesztőket, míg a kínálat csak 25,77%. A szűkösség felfelé viszi a fizetéseket, alacsonyabb a verseny, és több a lehetőség.
- A 2018-as változatban volt egy pont, ahol a munkaadók igényeiről tudhattál meg többet. Eszerint önmagadat is képezheted, mert a cégek lényegesen nagyobb hangsúlyt fektetnek a tapasztalatra (90.6%), a portfóliódra (72.7%), mint a végzettségedre. Megfelelő marketing mellett a tapasztalat megszerezhető azáltal, hogy saját projekteket fejlesztesz, amelyeket a portfóliódba teszel. Ez azt jelenti, hogy a JavaScript fejlesztővé válás legfőbb korlátozó tényezője te magad vagy. Senki másnak nincs irányítása a sorsod felett. Nincsenek befolyásos emberek, egyetemek, még szakmai gyakorlatra és tanúsítványokra (certifications) sincs szükséged. Saját tapasztalatom alapján Berlinben több tíz saját magát képző szoftverfejlesztővel dolgoztam együtt, akik több pénzt vittek haza, mint egy átlagos berlini. Sok más városra igazak ezek az elvek. Ha nem szeretnél otthonról elköltözni, akkor rendelkezésedre áll a távmunka, ami a 21. században már egy valós lehetőség. Külföldi színvonalon keresel, és ha megfelelően strukturálod az ügyfeleiddel való viszonyt, még a KATA adózás alapján az általad megkeresett pénz 90%-a nálad marad. Azaz több, mint havi nyolcszázezer forintot kereshetsz adózás után, miközben ezért jó eséllyel még heti negyven órát sem kell dolgoznod. Ha csak egy 30 eurós órabért nézünk, és átváltjuk 9000Ft-ra, kevesebb, mint 1350 óra alatt bruttó 12 millió forintot keresünk. Alkalmazottként jó eséllyel

kevesebb, mint évi 240 napot dolgozol. Ebből következik, hogy 30 eurós órabér mellett munkanaponként 5.625 órát kell dolgoznod.

- A TypeScript a JavaScript típusos változata. 2019-ben a Go, Kotlin, és Python után a TypeScript a negyedik helyen áll azok között a nyelvek között, amelyet a szoftverfejlesztők 2019-ben meg szerettek volna tanulni. Ez valószínű elhomályosította a JavaScript iránti keresletet, mert megoszlottak a válaszok a két nyelv között. Mi hozzátesszük, hogy a TypeScript ismeretéhez szükséges a JavaScript tudás is.

A 2019-es StackOverflow közvéleménykutatás hasonlóan bíztató eredményeket hozott:

- A Stackoverflow közönségében a JavaScript a legnépszerűbb nyelv, 67,8%-os részesedéssel. A HTML és a CSS áll a második helyen 63,5%-kal.
- Három JavaScript keretrendszer áll az összes elérhető keretrendszer élén: React.js (74,5%), vue.js (73,6%), és Express (68,3%).
- Bár az elérhető JavaScript fizetések alacsonyabbak, mint sok más területen, véleményem szerint ezek a számok félrevezetők. A megfelelő tudással rendelkező fejlesztők ugyanis egy kalap alá vannak véve az alacsonyabb tudású fejlesztőkkel. Ez sokkal inkább jellemző a JavaScript nyelvre, mint más nyelvekre. Saját tapasztalatom alapján nem ritka, hogy egy JavaScript első interjún a jelentkezők 70%-a elbukik. Nem azért, mert nehéz az interjú, hanem azért, mert annak a töredékét sem tudják, amit ebben a könyvben olvashatsz. Ezzel szemben egy C++ interjún sokkal többen átmennek az első interjún, mert nagyon kevesen táplálnak reményt egy megalapozatlan próbálkozásra, hiszen ha valamilyen csoda folytán fel is vennének egy képzetlen C++ fejlesztőt egy munkahelyre, pár héten belül bebizonyosodna, hogy alkalmatlan bármilyen használható munkára.

A fizetésekkel kapcsolatban javaslom, hogy végezz kutatást a payscale.com és a glassdoor.com oldalakon. Keress olyan JavaScript fejlesztői fizetéseket és sávokat, amelyek jellemzők arra a tudásra, amit elsajátítasz.

Bár a felvevőpiac számít, napjainkban nehéz a JavaScript megtanulásával rosszul járni. Emellett várható, hogy a JavaScript népszerűsége tovább nő.

> Összefoglalás:
>
> - A JavaScript és a JavaScript keretrendszerek nagy népszerűségnek örvendenek
> - A JavaScript fejlesztők iránti kereslet nagyobb, mint a kínálat

Miért könnyebb ma JavaScript fejlesztőnek állni, mint eddig bármikor?

A JavaScript fejlesztővé válás gazdasági hátterének áttekintése után levonhatjuk a következtetést, hogy mind kliens oldalon, mind szerver oldalon megfelelően nagy a kereslet. Ugyanakkor a világ összes pénze sem kárpótolna, ha a fejlesztői élmény rossz lenne.

A JavaScript nyelv elsajátítása ma könnyebb, mint eddig bármikor. Összehasonlításképp érdemes visszaemlékezni arra, hogy évekkel ezelőtt több frusztrációval kellett a JavaScript nyelvet megtanulni vágyó fejlesztőknek:

- Lassabb számítógépeken dolgoztunk

- A nyelv szabványa kiforratlanabb volt, amelynek következménye, hogy `if-else` ágakban különböző böngészők különböző megoldásait kellett használnunk, hogy működőképes kódot kapjunk,
- Az ES5 felhasználói élménye jelentősen rosszabb volt, mint az ES2015 és későbbi változatoké,
- A JavaScript nyelvet kevesen vették komolyan,
- A böngészőkben lévő fejlesztőkörnyezet és a hibakeresési eszközök fejletlenek voltak,
- Kevés nyílt forráskódú szoftver állt rendelkezésre, amelyek javarészt alacsony megbízhatósággal működtek,
- Nem voltak jó integrált fejlesztőkörnyezetek, vagy JavaScript fejlesztéshez megfelelő szövegszerkesztők.

A fenti pontokat leellenőrizheted, ha egy tapasztalt szoftverfejlesztővel beszélgetsz, aki tizenöt éve JavaScript programokat is írt. Azóta a JavaScript teljes átalakuláson ment át.

Ha problémába ütközöl, kérheted egy lelkes és megfelelő méretű nyílt forráskódú közösség segítségét. Sok projekt mögött ipari támogatás áll, pl. a Google és a Facebook is jó eséllyel megbízható JavaScript megoldásokat hoz létre. A szabványosítás folyamatának gyümölcseit 2015 óta éves rendszerességgel élvezhetjük. Megállapíthatjuk, hogy a JavaScript történetének legjobb korszakát éljük. És a jövőben további fejlődést várhatunk.

A JavaScript elsajátítása tudományos megközelítésben

Az első jó hírem, hogy azonnal elkezdhetsz kódot írni. A második, hogy másodperceken belül látod a programod futtatásának eredményét. Nem kell lefordítanod a kódot. Bármit csinálsz, a következményeket a képernyőn azonnal láthatod.

Nem azt állítom, hogy nagyon könnyű a JavaScript részleteit elsajátítani. Azt viszont mindenképp állítom, hogy a legtöbb nyelvnél gyorsabban kapsz visszajelzést a próbálkozásaid eredményéről. Ez miért is fontos?

Mert minden adott a flow élményhez (áramlatélmény), amelyet optimális élményként írnak le a tudományos kutatások. Optimális teljesítmény leadására alkalmas "zónában" érzed magad, és megszűnik az idő érzékelése. Annyira elmerülsz a feladatodban, hogy optimális tanulási élményben lesz részed.

Az áramlatélmény kialakulásának három szükséges feltétele van:

1. Tisztán látod a tevékenységed célját,
2. A feladat nem túl könnyű és nem túl nehéz,
3. Azonnali visszajelzést kapsz a cselekedeteid hatásáról.

Vizsgáljuk meg ezt a három feltételt részletesebben.

Tisztán látod a tevékenységed célját

Ha megvizsgálod a két hivatkozott közvéleménykutatást, láthatod, hogy van-e értelme JavaScript fejlesztőnek állni 2019-ben. Akár a tengerpartról is dolgozhatsz, és egy nemzetközi csapat része lehetsz. A JavaScript kód írásának élménye hasonlít a legózáshoz. És még jól is fizetnek érte. Sokan fél éven belül

produktív fejlesztőkké válnak. Ha ma még nem tudsz programozni, milyen más lehetőségeid vannak, amelyeket gyorsabban készpénzre tudsz váltani, mint a front end vagy full stack fejlesztővé válást?

A feladat nem túl könnyű és nem túl nehéz

A JavaScript elsajátítása nem egyértelmű folyamat. Több rejtvénnyel, csapdával, akadállyal találkozol a karriered során. Ugyanakkor ez nem kell, hogy akadályként lebegjen a fejed felett. Gondolkodj: kivé válnál akkor, ha soha nem kellene kihívásokkal megküzdened? Nem lennél semmire sem büszke az életedben. Ezek az akadályok érted vannak. Ezek az akadályok épp eléggé megnehezítik a dolgod ahhoz, hogy képes legyél áramlatélményben tanulni.

Véleményem szerint a JavaScript tanulása nem túl nehéz ahhoz, hogy áramlatélményben tanulj. Lehet ezzel a véleménnyel vitatkozni, hiszen elképzelhető, hogy egész életedben buszvezetőként dolgoztál, és életedben nem láttál még egy programozási nyelvet sem, és ebben az esetben aláírom, hogy lényegesen több alapozásra lesz szükséged. Amikor piacot kerestem az ES6 in Practice angol nyelvű könyvemnek, rájöttem, hogy nem érdemes a kezdő fejlesztőket megcéloznom, mert a könyv feltételezi, hogy legalább az alapokkal tisztában vagy. Tudnod kell, hogy hogyan kell programozni, mi az az `if` elágazás, mik azok a ciklusok, függvények. Néhány kezdő ügyfelem ki is fejezte, hogy nehezen haladt. Elsősorban az ő visszajelzésük alapján írtam meg ezt a könyvet.

Amint megérted az alapokat, készen állsz a haladóbb anyagok tanulmányozására, és megállíthatatlan leszel. Megfelelő iránymutatással fejlődni fogsz, anélkül, hogy frusztrálna, hogy túl nehéz a JavaScript elsajátítása.

Azonnali visszajelzést kapsz a cselekedeteid hatásáról

Ez a JavaScript nyelv egyik fő előnye. Amíg kliens oldalon maradsz és nem nehezíted meg túlságosan a saját dolgod, azonnali visszajelzést fogsz kapni az általad futtatott kódról. Alapesetben frissítened kell a böngésződet, de néhány eszköz már ezt a terhet is leveszi a válladról. Bármit is teszel, a kódod azonnal lefut, és látod az eredményét.

Éppen ezért a JavaScript tökéletes az áramlatélmény eléréséhez. Egyetlen más nyelvről tudok, amely majdnem hasonlóan jó feltételeket biztosít az alapok elsajátításához: ez a nyelv a Python.

Összefoglalás

Függetlenül attól, hogy első programnyelvedet tanulod, vagy csak új specializáció után nézel, fontos, hogy értékeld a JavaScript jelenlegi helyzetét.

Soha nem volt olyan jó a helyzet, mint napjainkban. A fejezet első felében átvettük a JavaScript nem mindig túl fényes múltját először a kliens oldalon. Ezt követte az egyértelműen gyorsabban sikeresebbé váló szerver oldali JavaScript bemutatása.

Az események sorozata egy olyan mértékű növekedést tett lehetővé, amelyet tíz éve nem tudtunk volna elképzelni. Rengeteg felhasználási területen érvényesülnek JavaScript fejlesztők, beleértve a virtuális valóság és blokklánc-technológiát. A JavaScript évről évre újabbnál újabb területeket hódít meg.

Mivel a JavaScript népszerűsége folyamatosan nő, a JavaScript fejlesztők jelene és jövője biztosított. Két közvéleménykutatás azt a következtetést vonta le, hogy nem járnak rosszul azok, akik ma a JavaScript nyelvre fogadnak.

Ha szeretnéd elsajátítani a JavaScript nyelvet, ennek a fejezetnek az utolsó része megállapítja, hogy a JavaScript tanulásának a feltételei közel optimális tanulási élményt biztosítanak. Mivel azonnali visszajelzést kapsz a próbálkozásaid eredményéről, amíg nem terheled túl magad, reális elvárás, hogy a tanulási időd meghatározó részét áramlatélményben töltsd, lényegesen gyakrabban, mint sok más programnyelv esetén.