Haladó JavaScript
PUSKASZETT

Top 3 JavaScript Interjúkérdés

1. CLOSURE

A **closure** egy kombináció: egy **függvény** plusz a **környezet,** vagy scope, amiben a függvényt létrehoztuk.

2. PROTOTÍPUSOS ÖRÖKLŐDÉS

A prototípusos öröklődés (prototypal inheritance) a JavaScript **öröklődési modellje**, amelyben az objektumok más objektumoktól örökölnek a **prototípusukon** keresztül.

ÖRÖKLŐDÉS

Az öröklődés (inheritance) egy mechanizmus, mely során **objektumokat készítünk** más objektumok alapján, amik ezeknek a szülő objektumoknak a tulajdonságait és metódusait öröklik.

3. THIS

JavaScriptben a függvények kiértékelésekor keletkezik egy **végrehajtási kontextus:** ez tárolja a környezetet, amelyben a függvény végrehajtása, meghívása megtörtént. A `this` egy **kulcsszó**, ami mindig **az aktuális végrehajtási kontextusra** mutat.

CodeBerry

Haladó JavaScript
PUSKASZETT

Scope a JavaScriptben

A JavaScriptben és általában a programozásban a scope a **végrehajtás aktuális kontextusa**. Ez az a környezet, amin belül az értékek és kifejezések "láthatók", használhatók lesznek.

SCOPE HIERARCHIA

A scope-ok **rangsorolhatók**: a gyermek scope-ok hozzáférnek a szülő scope-okhoz, de fordítva nem.

GLOBÁLIS SCOPE

A globális scope az a scope, ami minden más scope-ot **tartalmaz** és minden scope-ból **látható**.

```
var berry = "🍒";
```

FUNCTION SCOPE

A változók, amiket egy **függvényen belül** `var` kulcsszóval deklarálunk, függvényre scope-oltak lesznek, azaz **az őket körülvevő függvényben** érvényesek, ott **lokálisak.**

```
function fruitLogger () {
  var berry = "🍒";
  console.log(berry);
}
```

BLOCK SCOPE

A változók, amiket egy blokkon belül `let` vagy `const` kulcsszóval deklarálunk, blokkra scope-oltak lesznek, azaz **az őket körülvevő blokkon belül** érvényesek, ott **lokálisak**.

```
for (let fruit of fruitBasket) {
  const berry = "🍒";
  console.log(berry + fruit);
}
```

CodeBerry

Haladó JavaScript
PUSKASZETT

Hoisting
A JavaScriptben

A hoisting egy JavaScript mechanizmus, melynek során a változók és a függvénydeklarációk *úgy tűnnek, mintha* a scope-juk tetejére kerülnének a kód végrehajtása előtt.

A **változók** hoistolásakor csak a **deklaráció** mozdul el, az inicializáció nem.

A `let`tel vagy `const`tal deklarált változók nem inicializálódnak `undefined` értékkel a hoistoláskor, hanem **inicializálatlanok** maradnak.

A **függvények** hoistolásakor csak a **függvénydeklarációk** mozdulnak el, a függvénykifejezések nem.

Az **osztály**deklarációk és **osztály**kifejezések nem hoistolódnak, ezért nem használhatod az osztályokat a deklarálásuk előtt.

TIPP:

A hoisting problémák elkerülése érdekében mindig **a scope-juk tetején** deklaráld a változókat és a függvényeket.

CodeBerry

Haladó JavaScript
PUSKASZETT

this
A JavaScriptben

JavaScriptben a függvények kiértékelésekor keletkezik egy végrehajtási kontextus: ez tárolja a környezetet, amelyben a függvény végrehajtása, meghívása megtörtént.

A `this` egy kulcsszó, ami az aktuális végrehajtási kontextusra mutat.

A `this` értéke attól függ, hogyan hívjuk meg a kulcsszót tartalmazó függvényt.

INNEN HÍVJUK A FÜGGVÉNYT	ERRE MUTAT A `this`
globális kontextus	`Window`, másnéven **globális objektum**
egy objektum **metódusa**	az **objektum** amin meghívtuk
egy egyszerű **függvényhívás**	`Window`, másnéven **globális objektum**
egy függvény **strict mode**-ban	`undefined`
egy **eseménykezelő**	az esemény **célpontja**
egy **nyíl függvény**	a **szülő** kontextus
egy **konstruktor**	a létrehozott **objektum példány**
egy **osztály metódusa**	a létrehozott **objektum példány**

CodeBerry

Haladó JavaScript
PUSKASZETT

Kézműves `this` A JavaScriptben

A `call()` egy függvénymetódus, ami a függvényeket egy adott `this` értékkel **hívja meg.**
A függvényparamétereket **egyesével** várja.

Az `apply()` egy függvénymetódus, ami a függvényeket egy adott `this` értékkel **hívja meg.**
A függvényparamétereket egy **tömb** formájában várja.

TIPP:

A különbség a `call()` és az `apply()` között az, hogy milyen formában várják a továbbadandó függvényparamétereket: a `call()` egyesével, vesszővel elválasztva kér paramétereket, míg az `apply()` egy [tömböt] vár.

A `bind()` egy függvénymetódus, ami egy olyan **új függvényt hoz létre,** aminek meghívásakor a `this` kulcsszót a megadott értékre állíthatjuk be.

TIPP:

A `bind()` és a `call()` - `apply()` páros között az a különbség, hogy a `bind()` **későbbre** állítja be a `this` értékét, míg a `call()` és az `apply()` **meg is hívja** a függvényt.

CodeBerry

Haladó JavaScript
PUSKASZETT

Objektum-Orientált JavaScript

Az objektumorientált programozás (OOP) egy **programozási paradigma**, miszerint a programokat **objektumokba** rendezzük, modellezve a *dolgokat*, amikkel épp dolgozunk.

AZ **OOP** 3 ALAPFOGALMA

ABSZTRAKCIÓ

Ahol egyszerű **modell** készül egy komplex dologról, csak **a konkrét program céljainak fontos** dolgokra fókuszálva

ENKAPSZULÁCIÓ

Ahol ezt az egyszerű modellt egy komponens **becsomagolja**, csak adott módon engedve hozzáférést

ÖRÖKLŐDÉS

Ahol ezek a komponensek **használhatják** (öröklik) más szülő-komponensek tulajdonságait és metódusait

Az absztrakcióhoz és az enkapszulációhoz a JavaScriptben. **objektumokra** van szükség. Sablon alapján történő objektum-létrehozásra ezeket használhatod:

1. **factory függvényeket**
2. **konstruktor függvényeket** `new`-val
3. **osztályokat** `class`-szal

JavaScriptben a **prototípusos öröklődés** modell szerint öröklünk. Minden objektumból hivatkozás mutat a szülő objektumra amitől tulajdonságokat örökölhet.

CodeBerry

Haladó JavaScript
PUSKASZETT

JavaScript Design Patternek

A design patternek vagy tervezési minták gyakori szoftvertervezési problémák **újrahasznosítható megoldásai.**

A MODUL PATTERN egy **strukturális** tervezési minta, ami **enkapszulációt** ad egy IIFE és az abból visszaadott objektum literál segítségével.

A KONSTRUKTOR PATTERN egy **létrehozó** tervezési minta, ami az objektumok létrehozásában segít konstruktor függvények használatával.

A CLASS PATTERN egy **létrehozó** tervezési minta, ami az **ES6** `class` **szintaxist** hívja segítségül objektumok létrehozásához egy osztály sablon alapján.

A PUBLISH/SUBSCRIBE PATTERN egy **viselkedési** tervezési minta, ami egy eseménycsatornát (event channel) használ az objektumok közötti kommunikáció levezénylésére.

AZ MVC PATTERN egy **architekturális** tervezési minta, ami az alkalmazásokat egy adatkezelő Modellbe, egy felhasználói felületet építő Nézetbe és egy ezeket összekötő Kontrollerbe szervezi.

CodeBerry