**Válaszolj az alábbi kérdésekre minél részletesebben!**

1. **Mi a különbség a <span>, <div>, és az <article> elemek között?**

Mindegyikük a HTML dokumentum tagolására szolgál (szakaszok/containerek), de más szerepben.

* 1. A span inline elem, vagyis nem veszi fel a teljes sorszélességet. Szövegközi szakaszolásra szolgál (szöveg adott szakasza fogható meg fele). Önmagában nem ad hozzá látható formázást a befoglalt szövegrészhez, sem szemantikai jelentést.
  2. A div és az article blokk szintű elem, vagyis a teljes sorszélességet felveszik. A div a legáltalánosabb, míg az article , cikket, vagyis egy összefüggő tartalmi elemet hivatott magában foglalni, amely önmagában is értelmezhető (pl. fórumbejegyzés, blogbejegyzés, újságcikk). Elvárás, hogy az article-nek legyen valamilyen headingben (h1-h6) megadott címe.  
       
     Ennek szemantikus jelentősége van: a HTML5 filozófiájának megfelelően minden HTML elem megnevezése le is írja a szerepét, bár technikailag nincs közöttük különbség (= ugyanúgy használható egy div mondjuk „main-article” classal cikkek befoglalására, mint az article).   
       
     A megfelelő elemek használat a megfelelő szerepben szép, jól olvasható kódot biztosít (a Google számára is). A szemantikus elemek elnevezése azt a célt szolgálja, hogy olyan általános elemek, mint a div, végül ne is szerepeljenek a bildben (helyette header, footer, main, section, nav, etc.)

1. **Milyen különbségeket és hasonlóságokat ismersz a cookie-k és a localStorage között?**
   1. A fő különbség az, hogy a cookie-kat a szerver, míg a localStorage-ot a kliens tudja olvasni és kezelni.
   2. Ennek megfelelően a cookie mindig a HTTP kéréssel együtt kerül továbbításra, míg a localSorage-ben tárolt azonosító a kliens oldalon marad.
   3. A cookie-kban tárolható adatmennyiség jobban limitált: 4096 byte, míg a localStorage nagyobb, akár 5 MB adatot is tárolhat.
   4. Adatvédelmi/tárolási szempontból is eltérőek: a cookie-nak van lejárati ideje, és a cookie-k törlésével eltávolítható a böngészőből, míg a localStorage-ot a kliens tudja kiüríteni, lejárati ideje by default nincs, és csak cache ürítéssel törölheti a felhasználó.
2. **Milyen http metódusokat ismersz és melyiket milyen célra használják?**

A REST (Representational State Transfer) architektúrára épülő API-k az alábbi a felosztásban használják a legalapvetőbb HTTP kéréseket:

* GET: Adatlekérdezés, a szerver egy fejlécet és a bodyban található tartalmat adja át.
* POST: Adattovábbítás, a szerver fogadja és feldolgozza a request bodyban található tartalmat. Megfelelő értelmezéséhez be kell állítani a headerben a tartalomtípust.
* PUT: Adatmódosítás. Új adat tárolására is használható, de ha a kérésben hivatkozott adat már megvan a szerveren, azt módosítja a kérés.
* DELETE: Adat törlése.
* HEADER: HTTP header lekérdezése egy adott végpontról. Megegyezik a GET-tel, de a szerver nem ad vissza response body-t, csak headert.

A REST API-val szemben szemben a SOAP protokoll elsősorban POST requesteket használ.

Ezeken kívül létezik még CONNECT, TRACE, OPTION és PATCH lekérdezés is, de ezeket eddig nem kellett használnom az egyszerűbb webappok fejlesztése során, ezért csak felsorolás-szinten említem őket. (A PATCH részleges adatmódosítást végez, a CONNECT pedig a GET-hez hasonló, és úgy tudom, HTTPS alatt használandó.)

1. **Mi lesz az ’x’ értéke és miért?**

**var a = b = x = 0;**

**x = ++a + b++;**

Az érték 1.

Az első lépés 0-ra állítja minden változó kiinduló értékét.

Az összeadás során a ++a (pre-increment) először megnöveli a értékét 1-gyel, majd visszaadja azt. A b++ művelet (post-increment) először visszaadja b értékét, majd megnöveli 1-gyel, de így a-hoz még az eredeti érték kerül hozzáadásra.

Így a művelet kibontva: x = (1 + 0) + (0), vagyis x = 1.

A művelet végrehajtása után mindegyik változó értéke 1 lesz.

1. **Mi kerül az x változóba és miért?**

**var x = (a = 3);**

Az x változóba a 3 érték kerül. A végrehajtás során az a változó először megkapja a 3 értéket, majd x változó értékét az a-val megegyezőre állítjuk.

1. **Mit ír ki és miért?**

**var bar = function() { console.log("bar"); }**

**var foo = function() { console.log("foo"); }**

**console.log("baz");**

**setTimeout(foo, 1000);**

**setTimeout(bar, 500);**

A kiírt szöveg: baz, bar, majd foo, ebben a sorrendben. A baz azonnal, a bar 0,5, a foo pedig 1 másodperc késleltetéssel jelenik meg, mivel a console.log azonnal, a setTimeout funkcióval végrehajtott műveletek pedig a megadott 500, ill. 1000 ms késleltetéssel kerülnek végrehajtásra.

1. **Mi a különbség a két változó között?**

**var v1 = document.getElementById(’inputText’);**

**var v2 = $(’#inputText’);**

A getElementById metódus egy HTML collectiont állít elő, vagyis „kiemeli” a DOM-ból a megadott ID-val rendelkező elemet és minden child elemet, amely benne található.

A $ jellel megkezdett keresés egy JQuery objektumot ad vissza, ami NodeList típusú. Ennek előnye, hogy a JS array metódusok használhatók a benne foglalt elemek eléréséhez (pl. forEach, map, etc.)

1. **Mely mód(ok)on lehet elérni a „John Smith” szöveget tartalmazó változót?**

**var data = {**

**’person.name’: ’John Smith’**

**};**

Az itt látható képlet egy object literal. Kétféle notation révén érhető el a benne foglalt propertyk értéke:

- dot notation: data.key

- square bracket notation: data[key]

Mivel itt a hivatkozó kulcs stringként van megadva, ezért csak a square bracket notation használható:   
  
data['person.name'];

1. **Mit ír ki és miért?**

**var callbacks = [];**

**for (var i = 0; i < 5; ++i) {**

**callbacks.push(function() { console.log(i); });**

**}**

**callbacks[3]();**

5-öt.

A magyarázat: a callbacks egy array, amelybe 5x belehelyezünk (push) egy funkciót, amely az i értékét logolja ki. Végrehajtásra viszont csak ezek után kerül, ha a 3. elemet meghivatkozzuk, és a benne lévő callbacket meghívjuk, akkor az iteráláshoz használt i végső értékét, az 5-öt fogja kiírni.

(Egyszerűen megnézhető az összes callback által visszaadott érték mondjuk egy forEach segítségével:

callbacks.forEach(function(e) {

e();

});

Ilyenkor 5x kerül kilogolásra az 5.

)

1. **Mit ír ki és miért?**

**var a = 2, b = { c: 4 };  
var x = a, y = b;**

**x = 5;  
y.c = 6;**

**console.log(a + b.c);**

8. Az a értéke végig változatlanul kettő.

Az object literalként definiált b object viszont nem átadható érték, az y és a b változó is ugyanarra az objektumra mutat.

Így az y.c módosítása 6-ra állítja a b.c értékét is, vagyis a művelet során az a változó 2 értéke és b.c új, 6 értéke kerül összeadásra.

1. **Mi a különbség a két CSS selector között?**

**.element .symbol {}**

**.element.large .symbol {}**

Az első minden selector az „element” class „symbol” classba tartozó leszármazottat választja ki.

A második selector csak az „element” és „large” classal egyaránt rendelkező elemek „symbol” classba tartozó leszármazottait választja ki.

1. **Milyen színű lesz a felirat és miért?**

**<div class="form-square">**

**<div class="seven-col">**

**Hello World!**

**</div>**

**</div>**

**div.form-square > div {**

**color: blue;**

**}**

**.seven-col {**

**color: red;**

**}**

Kék. Ugyan a seven-col classú divnek van saját color propertyje, de a szülőre vonatkozó beállítás (a form-square minden div childja kék font színnel rendelkezzen) magasabb specificity értékkel bír, ezért felülírja a child div saját osztályához tartozó beállítást.

1. **Adatbázis tervezési feladat**

**Tervezd meg egy webáruház adatbázis sémáját – relációs adatbázisban (táblákban gondolkodj)!**

**Termékeket szeretnénk tárolni úgy, hogy az adminok a felületen vehessék fel a termékeket. A termékhez bármennyi, különböző típusú attribútum tartozhat. Új terméktípus, vagy új termék-attribútum felvételéhez ne kelljen sémát módosítani!**

**Példa a termékekre és attribútumaikra:**

**Dell Vostro X500  
- memória: 16gb   
- processzor magok száma: 4   
- bevezetés ideje: 2015-12-31**

**Braun A1800 hajszárító  
- szín: fekete, fehér  
- erősség: 1800W  
- hideglevegő fújás: igen   
- garanciális hónapok száma: 12**

**A cél tehát olyan adatstruktúrát kialakítani, hogy ha ezentúl notebookoknál a kijelző felbontását is tárolni szeretnénk, akkor ehhez ne kelljen az adattáblák struktúráját módosítani. Ha több alternatív megoldást is találsz, írhatsz többet is. Az adatstruktúrát bármilyen formában leírhatod – csak értsük meg **

A webáruház termékeket, ha fix az adatstruktúra, úgy lenne logikus tárolni, hogy egy táblában minden sor egy termék, az oszlopok pedig az egyes jellemzőket írják le.

1. Megoldás: külön táblák, rugalmas sormennyiséggel

Jelen esetben azonban nem módosíthatjuk a táblákat, illetve azok oszlopait az adatbázison belül, ezért egy táblának az állandó paramétereket kell tartalmaznia, ebben továbbra is egy sor egy terméket tartalmaz.

Egy másik tábla viszont univerzális oszlopokat tartalmaz: a primary key a termék cikkszáma, az egyik tároló oszlop a kulcs, a másik pedig a hozzá rendelt érték. Így egy adott termékhez tetszőleges mennyiségű sor tartozhat a szükséges tárolt adatoknak megfelelően, amelyek a cikkszám alapján kereshetők.

(Megj.: az itt vázolt sémák még a pontos oszlop elnevezéseket tartalmazzák, és az adattípusoknak is csak a fő kategóriáját határoztam meg – a szükséges karakterhosszokat a termékek/best practice ismeretében lehetne kialakítani, az INT értékeket pedig, ha soha nem szeretnénk a táblához nyúlni, nem akartam korlátozni. Pl. gyártó neveknél jó, ha megvan a teljes név, ami rendszerint nem túl hosszú, de kategóriáknál/alkategóriáknál így „vakon” csak az a foolproof, ha van elég helyem arra is, hogy „egér”, vagy „notebook adapter”, de arra is, hogy „informatikai zabhegyező szervizkészlet fluxuskondenzátor talpazat kiegészítők”. Igaz, egy VARCHAR(255) valószínűleg bármire elég.)

Táblák és oszlopok:

1. Termékek állandó paraméterei
   1. Cikkszám (INT) (primary key)
   2. Gyártó (CHAR)
   3. Modell név (CHAR)
   4. Termék főkategória (pl. számítástechnika, háztartási gép, szórakoztató elektronika) (CHAR)
   5. Termék alkategória (pl. notebook, monitor, hajszárító) (CHAR)
   6. Ár (INT)
   7. Készlet (INT)
   8. stb.
2. Műszaki paraméterek
   1. Cikkszám (INT) (primary key)
   2. Paraméter kulcs (pl. „Felbontás”) (CHAR)
   3. Paraméter érték (pl. 1080p) (CHAR)
3. Műszaki paraméter validáció
   1. Sorszám (INT) (primary key – itt minden más ismétlődhet, ezért kell vmilyen arbitrális egyedi ID)
   2. Termék alkategória (CHAR)
   3. Paraméter kulcs (CHAR)
   4. Paraméter érték (CHAR)

A paraméter kulcsokat és értékeket ez esetben egyaránt valamilyen szöveges formátumban tárolnám, mivel sokféle formátum szóba jöhet. Pl. Kamera felbontás lehet 5 megapixel, ekkor elég lenne egy INT típusú mező, de a képernyőfelbontás már két adat is lehet, pl. 1280x720, ez pedig már szöveg.   
  
Az adatbevitelnél ezért érdemes a szóba jöhető termékekhez validációs referencia-táblát is készíteni, hogy pl. az adatbevitelnél egy dropdown listából, vagy prediktív javaslatokból csak létező értékeket lehessen kiválasztani, és ne kelljen mindig manuális adatbevitelre hagyatkozni. Ebben a táblában (3.) a pirmary key a termék alkategória oszlop, amely leírja, hogy milyen paraméterekkel bírhat egy notebook és egy hajszárító (képernyőfelbontás vs. Hideg levegő fújás boolean (TINYINT)).

A táblában ahány értéke lehet egy adott termék paraméter kulcsnak, az adott kulcs annyi sorban szerepel, eltérő értékkel.

2. Megoldás: MySQL JSON adattípus használata

A rögzített oszlopokkal használt első táblázat-változat mellett használható a MySQL JSON típusú adatoszlopa is, mivel a JSON rugalmasan változtatható kulcs-érték párokat tárol. Így a megjeleníteni kívánt termékjellemzők egy JSON típusú oszlopba kerülnek, és ezen belül kell szerepelnie a megfelelő – éppen használt – kulcs-érték pároknak.

Tehát a tábla oszlopai:

* 1. Cikkszám (INT)
  2. Gyártó (CHAR)
  3. Modell név (CHAR)
  4. Termék főkategória (pl. számítástechnika, háztartási gép, szórakoztató elektronika) (CHAR)
  5. Termék alkategória (pl. notebook, monitor, hajszárító) (CHAR)
  6. Ár (INT)
  7. Készlet (INT)
  8. **Műszaki paraméterek (JSON)**

Így csak az érintett, jelen esetben a 8. oszlop JSON állományának kulcs-érték párosait kell módosítani.

Ehhez a táblához a JSON formátumnak hála, már tömörebben lehet validációs táblát készíteni, amely az alábbi oszlopokból épül fel:

1. Termék alkategória (CHAR) (primary key)
2. Műszaki paraméterek (JSON)

Ez a fajta validáció (vagy az előző) több tárolt adattípusra is kiterjeszthető, pl. gyártók, vagy termék kategóriákon belüli lehetséges alkategóriák, etc.