

# WEBFEJLESZTÉS HAGYOMÁNYOS ESZKÖZÖKKEL

6. forduló

TRAINING360

A kategória támogatója: Training360 Kft.

## Ismertető a feladathoz

Iszogatód a cégnél a kávéd, van még majd 30 perced kezdésig, mikor kedvesen mosolyogva eléd raknak egy tesztet. Jelentkeztél, hogy feljebb léphess a tapasztalati (fizetési) ranglétrán, és bizonyítanod kell. Több fordulóban. 25 kérdés. 25 perc elég lesz? Aztán stand up. Így megy ez. No comment.

Felhasznált idő: 00:00/25:00

Elért pontszám: 0/25

## 1. feladat 0/1 pont

Válaszd ki a safe HTTP metódust! (RFC szerint)

### Válasz

☒ GET

Ez a válasz helyes, de nem jelölted meg.

☐ POST

☐ PUT

☐ PATCH

☐ DELETE

### Magyarázat

Akkor *safe* a metódus, ha nem változtatja meg a szerver állapotát. Ez pedig egyedül a *GET*-re igaz, hiszen a többi az erőforrásokat módosítja.

[Hypertext Transfer Protocol -- HTTP/1.1](#)

## 2. feladat 0/1 pont

Válaszd ki az idempotens HTTP metódusokat! (RFC szerint)

### Válaszok

☒ GET  
Ez a válasz helyes, de nem jelölted meg.

☐ POST

☒ PUT  
Ez a válasz helyes, de nem jelölted meg.

☐ PATCH

☒ DELETE  
Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

Idempotens egy művelet, ha többször is végrehajtva ugyanazt az eredményt kapjuk. Azaz jelen esetben mindegy, hogy egyszer vagy N-szer ( $N > 0$ ) kerül elküldésre a kérés. Ennek a megvalósítására nekünk is figyelniünk kell, mert ha szerveroldalról egy DELETE kérésre egyszer 200-at, aztán 404-et kapunk, akkor nem idempotent.

[Hypertext Transfer Protocol – HTTP/1.1](#)

## 3. feladat 0/1 pont

Válaszd ki, melyik állítások igazak a HTTP GET metódusra!

### Válaszok

☒ Cache-elhető a válasz  
Ez a válasz helyes, de nem jelölted meg.

☒ Könyvjelzőbe elmenthető  
Ez a válasz helyes, de nem jelölted meg.

☐ Böngészőben küldés után, az oldal frissítésekor újra el lesznek küldve az adatok

☒ Kevésbé biztonságos, mint a POST  
Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

- A GET metódus erőforrást kér el, ezért nagyon egyszerűen cache-elhető
- Mivel az URL-hez fűzi hozzá az adatokat, ezért az URL-t könyvjelzőbe elmenthetjük
- A POST-al ellentétben nem fogja újra elküldeni a kérést egy refresh után
- A nem ASCII karakterek ki lesznek cserélve egy %HEXA formátumú kódra
- Mivel az URL-ben van az összes adat, a jelszót is beleértve, ezért kevésbé biztonságos, mint a POST

**Kedves Versenyzők!**

**A "Csak ASCII karaktereket támogattak" válaszlehetőséget töröltük, ugyanis nem egyértelmű annak helyessége, vagy helytelensége (a mostani RFC szabvány szerint lehetőség van GET metódus esetében is body-t küldeni).**

## 4. feladat 0/1 pont

Milyen adatszerkezetet használnál az alábbi életszerű példák implementálása során?:

Mozgólépcső, Spotify (vagy egyéb lejátszó) next-back gombok, Windows-os Alt + Tab

### Válasz



Linked list

Ez a válasz helyes, de nem jelölted meg.



Graph



Hash Tables



Queue



Stack



Tree

### Magyarázat

Egy Linked list (Láncolt lista) elemek sorozatából épül fel, és minden tartalmaz egy hivatkozást a következő elemre. Amennyiben az utolsó elem tartalmazza a hivatkozást az első elemre, úgy megvalósul a körkörösség, Tehát a lejátszó esetében, vagy az Alt + Tab nyomkodásánál az utolsó elemet elérve újra az elsőre ugrunk.

## 5. feladat 0/1 pont

Milyen adatszerkezetet használnál az alábbi életszerű példák implementálása során?:

Browser history, log, undo/redo, emailek

### Válasz



Linked list



Graph



Hash Tables



Queue



Stack

Ez a válasz helyes, de nem jelölted meg.



Tree

### Magyarázat

A Stack (Verem) egy LIFO (Last In First Out) adatszerkezet. A legkésőbbi elemhez férünk hozzá először. Jönnek az újabb és újabb emailek, belekerülnek a Verembe, mindig a legújabb van elől.

## 6. feladat 0/1 pont

Milyen adatszerkezetet használnál az alábbi életszerű példák implementálása során?:

HTML DOM, Domain Name Server, Fájltrendszer, IP routing tábla

### Válasz



Linked list



Graph



Hash table

☐ Queue

☐ Stack

☒ Tree

Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

Ha csak a DOM Tree-re gondolunk, az elemek fa struktúrában helyezkednek el, Van egy html gyökér elem, felállítható a szülő-gyermek kapcsolat, vannak szomszédos elemek, stb.

## 7. feladat 0/1 pont

Milyen adatszerkezetet használnál az alábbi életszerű példák implementálása során?:

Virtual DOM, Repülési hálózatok, Userok és kapcsolódásaik a social média oldalakon, legrövidebb útvonal kiszámítása GPS-eknél

### Válasz

☐ Linked list

☒ Graph

Ez a válasz helyes, de nem jelölted meg.

☐ Hash table

☐ Queue

☐ Stack

☐ Tree

## Magyarázat

Ha csak egy social media oldalra gondolunk, minden user egy-egy pont, közöttük a kapcsolatot egy-egy vonallal ha berajzoljuk kapunk egy csodás gráfot.

## 8. feladat 0/1 pont

Válaszd ki, azokat a rövidítés-fogalom párokat, ahol az adott rövidítés valóban a mellette lévő fogalmat takarja!

### Válaszok

☒ BDUF - Big Design Up Front

Ez a válasz helyes, de nem jelölted meg.

☒ CDN - Content Delivery Network

Ez a válasz helyes, de nem jelölted meg.

☐ SOC - Single Origin Content

☒ MVP - Minimum Viable Product

Ez a válasz helyes, de nem jelölted meg.

☐ POC - Proof of Certification

☐ RAIL - Request, Animation, Idle, Lazy

☒ FOIT - Flash of invisible text

Ez a válasz helyes, de nem jelölted meg.

☐ FOIT - Flash of invisible text

## Magyarázat

A rövidítések helyes megoldásai a következők:

- BDUF - Big Design Up Front
- CDN - Content Delivery Network
- SOC - Separation Of Concerns
- MVP - Minimum Viable Product
- POC - Proof Of Concept
- RAIL - Response, Animation, Idle, Load
- FOIT - Flash of invisible text
- FOUT - Flash of unstyled text

## 9. feladat 0/1 pont

Adott az alábbi HTML és JS kód:

```
1 <input type="file" id="file" name="file" />
2
3 <script>
4   const fileInput = document.querySelector('#file');
5
6   fileInput.addEventListener('change', (event) => {
7     console.log(event.target.files[0].type);
8   })
9 </script>
```

Saját gépen egy *.jpeg* kiterjesztésű képfájl kiterjesztését átírom pdf-re.

Betellőzöm ezt a fájlt a fenti alkalmazás segítségével.

Mi lesz a konzolon a kód lefutása után?

### Válasz

☐ image/jpeg

☒ application/pdf  
Ez a válasz helyes, de nem jelölted meg.

☐ Error

☐ image/jfif

## Magyarázat

A *type* nem néz bele a fájlba a MIME típus megállapításához, hanem a fájl kiterjesztése alapján kalkulálja ki azt. Emiatt hiába van egy kép fájlunk, mivel a kiterjesztést módosítottam, ezért *application/pdf* lesz a konzolon. Soha nem elég csak kliensoldalon ellenőrizni a fájl típusát.

## 10. feladat 0/1 pont

Az alábbiak közül melyik operator, illetve statement használható JavaScriptben?

### Válaszok

- ☒ with statement  
Ez a válasz helyes, de nem jelölted meg.
- ☒ label statement  
Ez a válasz helyes, de nem jelölted meg.
- ☐ goto statement
- ☒ in operator  
Ez a válasz helyes, de nem jelölted meg.
- ☐ of operator
- ☒ void operator  
Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

A goto statement nem elérhető JavaScriptben, de a label igen, ahogy a break, és a continue is.

Az of operátor önállóan nem létezik, de for of ciklus igen.

A többi operator és statementek létezik JavaScriptben, de ilyen-olyan okokból kifolyólag nem használjuk.

[with](#) [label](#) [in](#) [void](#)

## 11. feladat 0/1 pont

Válaszd ki azokat az állításokat, melyek igazak az Observer tervezési mintára!

### Válaszok

- ☒ Az Observer esetében a megfigyelők és a megfigyeltnek tudnak egymásról  
Ez a válasz helyes, de nem jelölted meg.
- ☒ Az Observer általában szinkron módon van megvalósítva  
Ez a válasz helyes, de nem jelölted meg.
- ☐ Az Observer egy Creational Design Patterns
- ☒ Az Observer estében a komponensek szoros kapcsolódnak (tight coupling)  
Ez a válasz helyes, de nem jelölted meg.
- ☒ Az Observer egy Gang of Four tervezési minta  
Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

Az Observer és a PubSub minta hasonlóak, de nem ugyanazok. Természetesen mind a kettő Behavior Design Patterns.

A különbségek röviden összefoglalva:

- Az Observer Gang of Four minta a PubSub nem
- A Observer általában szinkron az PubSub aszinkron van megvalósítva
- Míg az Observer esetében szorosan kapcsolódnak a komponensek a PubSub esetében lazán, egy harmadik félen keresztül történik a kommunikáció

## 12. feladat 0/1 pont

Válaszd ki azokat az állításokat, melyek igazak a PubSub tervezési mintára!

### Válaszok

- ☒ A PubSub minta gyakran használt elosztott rendszereknél a kommunikáció során  
Ez a válasz helyes, de nem jelölted meg.
- ☒ A PubSub általában aszinkron módon van megvalósítva  
Ez a válasz helyes, de nem jelölted meg.
- ☒ A PubSub nem követeli meg, hogy a megfigyelők és a megfigyelték tudjanak egymásról  
Ez a válasz helyes, de nem jelölted meg.
- ☒ A PubSub estében a komponensek lazán kapcsolódnak (loose coupling)  
Ez a válasz helyes, de nem jelölted meg.
- ☐ A PubSub egy Gang of Four tervezési minta

## Magyarázat

Az Observer és a PubSub minta hasonlóak, de nem ugyanazok. Természetesen mind a kettő Behavior Design Patterns.

A különbségek röviden összefoglalva:

- Az Observer Gang of Four minta a PubSub nem
- A Observer általában szinkron az PubSub aszinkron van megvalósítva
- Míg az Observer esetében szorosan kapcsolódnak a komponensek a PubSub esetében lazán, egy harmadik félen keresztül történik a kommunikáció

## 13. feladat 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?



```
1 let arr;  
2 for (var i = 5; i > 0; i--) {  
3   arr.push(i);  
4 }  
5 console.log(arr);
```

### Válasz

☐ [1, 2, 3, 4, 5]

☐ [5, 4, 3, 2, 1]

☒ Error  
Ez a válasz helyes, de nem jelölted meg.

☐ undefined

### Magyarázat

Az arr változónak nem adtunk kezdőértéket, így undefined lesz, és mint ilyen nincs *push()* metódusa, ezért *Error*-t kapunk.

## 14. feladat 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?



```
1 const fruits = {  
2   dragonfruit: {  
3     name: 'dragonfruit',  
4     price : 10  
5   }  
6 };  
7  
8 const { dragonfruit: { name: n = 'sárkánygyümölcs' } } = fruit;  
9  
10 console.log(n);
```

### Válaszok



- ☐ undefined
- ☐ sárkánygyümölcs

☒ dragonfruit  
Ez a válasz helyes, de nem jelölted meg.

☒ Error  
Ez a válasz helyes, de nem jelölted meg.

## Magyarázat

Egy nested destructuring volt használva.

A *fruits* objectból lekérdezzük az *dragonfruit* property-t, majd annak a *name* tulajdonságát *n* alias megadva neki és az sárkánygyümölcs default value-t.

**A feladatkiírásban volt egy elírás, ezért az "error" válaszlehetőséget is elfogadjuk.**

## 15. feladat 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?

```
1  const [first] = ['Éva', 'Bendegúz', 'Ádám'].sort();  
2  
3  console.log(first);
```

### Válasz

- ☐ Éva
- ☒ Bendegúz  
Ez a válasz helyes, de nem jelölted meg.
- ☐ Ádám
- ☐ Error

## Magyarázat

A *sort()* metódus az UTF-16 kódok alapján rendez. Abban pedig az "Á" és az "É" betű az angol karakterek után kullog, tehát "Ádám" és "Éva" marad a tömb végén. Mivel az első elem "Bendegúz" lesz, és destructuringgal az első elemet vesszük ki a tömbből, így a konzolon a "Bendegúz" string fog megjelenni.

## 16. feladat 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?



```
1  const numbers = [];  
2  for (var i = 0; i < 4; i += 1); {  
3      numbers.push(i + 1);  
4  }
```

### Válasz

☐ [1, 2, 3]

☒ [5]

Ez a válasz helyes, de nem jelölted meg.

☐ [1, 2, 3, 4]

☐ []

☐ [4]

### Magyarázat

A trükk abban rejlik, hogy a for záró zárójele után volt egy pontosvessző, tehát üres volt a ciklusmag.

Emiatt a `numbers.push()` csak egyszer fut le, mikor már a for ciklus futása végzett.

Mivel az `i` ciklusváltozó a `var` kulcsszóval lett létrehozva, ezért a cikluson kívül is elérhető, értéke a kilépési feltételnek megfelelő érték, azaz 4. A tömbben így egy érték lesz a `4+1`, azaz 5.

**Kedves Versenyzők!**

**A feladatban szándékosan volt a pontosvessző a for loop végén.**

**Ugyan a `console.log()` kimaradt, de a választható opciók közül mindegyik egy tömb volt, így egyértelmű, hogy a `numbers` értékét kellett volna kiíratni.**

**17. feladat** 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?

```
1  const creature = Object.create({
2    name: 'Winnie-the-Pooh',
3    age: 96
4  });
5
6  delete creature.age;
7
8  console.log(creature);
```

### Válasz



{ }

Ez a válasz helyes, de nem jelölted meg.



{ name: 'Winnie-the-Pooh', age: 96 }



null



{ name: 'Winnie-the-Pooh' }

### Magyarázat

Az `Object.create()` egy meglévő objektumot használ fel, mint prototípust. Azaz nem a *creature* hanem a *creature* prototípusa tartalmazza a `name` és `age` tulajdonságokat. Mivel a *delete*-el a *creature* `age` tulajdonságát akarjuk törölni, de közvetlenül nincs neki, így nem is történik semmi. Maga a *person* pedig továbbra is egy empty object marad.

Természetesen a prototípust lekérhetjük, és látszódnak a tulajdonságok: `Object.getPrototypeOf(person)`

**18. feladat** 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?

```
1 function* generate() {  
2   yield 1;  
3   yield 2;  
4   return 3;  
5 }  
6  
7 console.log([...generate()]);
```

#### Válasz

☐ [1]

☐ [3]

☒ [1, 2]

Ez a válasz helyes, de nem jelölted meg.

☐ [1, 2, 3]

#### Magyarázat

A *generate()* meghívása egy *Generator*-t ad vissza.

Mivel ez iterálható object a spread operátor segítségével kibontjuk, és egy tömbbe tesszük az elemeket.

A spread nem a teljes objektumot, hanem csak a *value*-kat helyezi el a tömbben.

A spread egészen addig megy, a *done* értéke *true* nem lesz, tehát a *return* utáni érték már nem fog belekerülni.

[Link](#)

#### 19. feladat 0/1 pont

Mi lesz az alábbi kód lefutása után a konzolon?



```
1  const obj = {  
2    valueOf() { return 42 },  
3    toString() { return 'forty-two'},  
4  };  
5  
6  console.log(String(obj) + obj + `${obj}` + (obj));
```

### Válasz

- ☐ 42424242
- ☐ 4242forty-two42
- ☐ 84-twoforty-two42

☒ forty-two42forty-two42  
Ez a válasz helyes, de nem jelölted meg.

### Magyarázat

A `String` constructor a `toString()` metódust használja, ugyanúgy, mint a template string is, míg a `+` a `valueOf()`-ot használja a háttérben.

Tehát:

- `String(obj) = forty-two`
- `(String(obj) + obj) = forty-two42`
- `String(obj) + obj + `${obj}` = forty-two42forty-two`
- `String(obj) + obj + `${obj}` + (obj) = forty-two42forty-two42`

[Link](#)

## 20. feladat 0/1 pont

Melyik függvény fog `Error`-t dobni (nem fut le a `catch` ág kódja) ha meghívjuk?



```
1  async function returnWithAwait () {  
2    try {  
3      return await Promise.reject(new Error());  
4    } catch (error) {  
5      return 'OK!';  
6    }  
7  }  
8  
9  async function returnWithoutAwait () {  
10   try {  
11     return Promise.reject(new Error());  
12   } catch (error) {  
13     return 'OK!';  
14   }  
15 }
```

### Válasz

☐ returnWithAwait()

☒ returnWithoutAwait()

Ez a válasz helyes, de nem jelölted meg.

☐ Mindkettő

☐ Egyik sem

### Magyarázat

Ez azon kivétel, amikor van különbség a *return* és a *return await* között.

A *returnWithAwait()* esetében az *await* miatt megvárja, míg rejected lesz a Promise. Mivel hibát dobott, ezért a *catch* ágra ugrik. A Promise resolved lesz az 'OK' értékkel. A *returnWithoutAwait()*, mivel nincs ott az *await*, nem várja meg a futást, és Error-t dob. Tehát csak akkor ugrik a catch ágra, ha ott az *await*.

## 21. feladat 0/1 pont

Jelöld be a helyes állításokat!

### Válaszok

☒ Az olyan unicode karaktereket, mint a "👨" felhasználható, mint className:



```
1 <div class="💩">Mr. Hankey, the Christmas Poo</div>
```

Ez a válasz helyes, de nem jelölted meg.



Használhatok inline stílusmegadásnál css variable-t az alábbi módon:



```
1 <div class="timmy" style="--textcolor: red;">Timmy Timmy Timmy Timmy...</div>
```

Ez a válasz helyes, de nem jelölted meg.



A transform: *translate(50%, 50%)* helyett írhatom azt, hogy: *translate: 50% 50%*

Ez a válasz helyes, de nem jelölted meg.



A natív CSS *@apply* már széleskörben támogatott a böngészők körében

## Magyarázat

Amennyiben Unicode karakterkódolást használunk (mi más?) használhatjuk a "💩" és egyéb unicode karaktereket akár classname-ben vagy bárhol.

Inline stílusmegadás esetében ugyanúgy elérjük a változókat, mintha beágyazva, vagy belinkelve, és ezek JavaScriptből is lekérdezhetők a styles propertyn keresztül. Pl.: `document.querySelector('.timmy').style.getPropertyValue('--textcolor')`

A különböző transzformációk külön propertyként történő megadása már a legtöbb böngészőben támogatott, így nem kell mindig megadni a *transformot*. [Translate - caniuse](#)

Az *@apply* rule sajnos még csak proposalban van: [@apply - caniuse](#)

## 22. feladat 0/1 pont

Az alábbiak közül melyik a legspecifikusabb szelektor?

### Válasz



nav .mainMenu > a:hover::before



nav .mainMenu > ul li a:hover::before



nav .mainMenu > ul li.active a:hover::before

Ez a válasz helyes, de nem jelölted meg.



.nav .mainMenu > ul li a:hover::before

## Magyarázat

CSS specificity kalkulátor alapján:

`nav .mainMenu > a:hover::before`

0 IDs

2 Classes, attributes and pseudo-classes

3 Elements and pseudo-elements

+ Duplicate

`nav .mainMenu > ul li a:hover::before`

0 IDs

2 Classes, attributes and pseudo-classes

5 Elements and pseudo-elements

+ Duplicate

`nav .mainMenu > ul li.active a:hover::before`

0 IDs

3 Classes, attributes and pseudo-classes

5 Elements and pseudo-elements

+ Duplicate

`.nav .mainMenu > ul li a:hover::before`

0 IDs

3 Classes, attributes and pseudo-classes

4 Elements and pseudo-elements

+ Duplicate

## 23. feladat 0/1 pont

Adott a következő html kód:

```
1 <div class="red green">Color</div>
```

és 3 CSS kód:

A.





```
1  .green {  
2    color: green;  
3  }  
4  
5  .red {  
6    color: red;  
7  }
```

B.



```
1  .red {  
2    color: red;  
3  }  
4  
5  .green {  
6    color: green;  
7  }
```

és C.



```
1 .red.red {  
2   color: red;  
3 }  
4  
5 .green {  
6   color: green;  
7 }
```

Milyen színű lesz sorban, **A**, **B**, és **C** kódok esetén a *díven* belül a szöveg?

#### Válasz

- ☐ piros, zöld, zöld
- ☒ piros, zöld, piros  
Ez a válasz helyes, de nem jelölted meg.
- ☐ zöld, piros, zöld
- ☐ zöld, piros, piros

#### Magyarázat

Az **A** és **B** esetben amelyik kód később van a CSS-ben az fog érvényesülni, tehát piros, zöld.

A **C** esetben a specifikusság miatt piros lesz a szövegszín.

## 24. feladat 0/1 pont

Az alábbiak közül melyik **nem** egy szabvány szerinti media query típus?

#### Válasz

- ☒ projector  
Ez a válasz helyes, de nem jelölted meg.
- ☐ braille
- ☐ aural
- ☐ speech

#### Magyarázat

A szabványos media queryk:

- all

- print
- screen
- tty
- tv
- projection
- handheld
- braille
- embossed
- aural
- speech

[Media Queries Level 4](#)

## 25. feladat 0/1 pont

Hogy érzed, ennyi kérdés ennyi idő alatt elég volt? Netán sok(k)? Esetleg kevés?

### Válaszok

- ☒ Csak ennyi? Kérek még!  
Ez a válasz helyes, de nem jelölted meg.
- ☒ Elég volt, köszí.  
Ez a válasz helyes, de nem jelölted meg.
- ☒ Ez elég erőstre sikerült.  
Ez a válasz helyes, de nem jelölted meg.
- ☒ Na elmész a...  
Ez a válasz helyes, de nem jelölted meg.
- ☐ "Az ötöst szádra ne vedd..." azaz nem érdekel az ingyen egy pont - sem a gyilkos nyúl - , a véleményem megtartom magamnak.

### Magyarázat

Az most nincs. :)



[Legfontosabb tudnivalók](#) [Kapcsolat](#) [Versenyszabályzat](#) [Adatvédelem](#)

© 2023 Human Priority Kft.

KÉSZÍTETTE **cone**

Megjelenés

Világos