**ACTIVITAT AVALUABLE AC2**

**Mòdul:** MP06- Desenvolupament web en entorn client

**UF:** UF2 – Estructures definides pel programador

**Professor:** Albert Guardiola

**Data límit d’entrega:** 2/12/2024 23:59

**Mètode d’entrega:** Per mitjà del Clickedu de l’assignatura. Les activitats entregades més enllà de la data límit només podran obtenir una nota de 5.

**Instruccions:** S’ha d’entregar un únic document amb el nom:

***MP06-UF2-AC1-Nom\_Alumne.pdf***

Es valorará la presentació.

**Tasca 1**. Des d’un script principal de JS (*script.js*) volem carregar un segon script (*1.js*), per fer servir la funció *funcion1* des de l’script principal.

Per carregar l’script secundari (*1.js*) podem fer servir la següent funció *loadScript*, que crea un element HTML amb el tag ‘script’ i configura el seu atribut ‘src’, abans d’afegir-lo al DOM.

function loadScript(src) {

    let script = document.createElement('script');

    script.src = src;

    document.head.append(script);

}

1. A l’script principal, prova de cridar *funcion1* just després de la càrrega de l’script.

loadScript('1.js');

funcion1();

Quin problema observes? Per què ocorre?

→ La funcion loadScript(‘1.js’) tarda en cargarse y si llamas a la funcion1() antes que se termnie de cargar se la funcion loadScript nos dara error por que la funcion aun no existe.

1. Modifica la funció *loadScript* perquè admeti un callback, de manera que sigui ella mateixa qui s’encarregui de cridar *funcion1* quan l’script secundari ja estigui carregat. Per assegurar-nos que, en el moment de la crida del callback, l’script no només està afegit al DOM sinò que també està llegit i carregat en memòria, ell callback s’haurà d’executar quan es produeixi l’event ‘load’ de l’script: per això, hauràs d’afegir un listener a l’element ‘script’ i posar-li el callback com a *handler*. La crida a *loadScript* haurà de passar-li *funcion1* com a callback.

loadScript('1.js',funcion1);

Quin problema observes? Per què ocorre?

→ L

1. Com que *funcion1* no existeix encara en memòria quan executem loadScript, enlloc de passar a loadScript el callback *funcion1*, passa-li una funció anònima, sense paràmetres, en notació de funció fletxa, que cridi en el seu cos a *funcion1*. Ara ja hauries de poder executar funcion1 sense problemes.

loadScript('1.js',()=>funcion1());

1. Modifica la funció *funcion1* perquè retorni un número aleatori entre 0 i 9. Modifica l’script principal perquè es reculli aquest valor i es mostri per consola el missatge “La función 1 ha devuelto el número x”.
2. Volem que, si el paràmetre ‘src’ és una cadena buida, la funció *loadScript* informi d’un error. Modifica la funció perquè admeti un errorCallback que s’executi a dins de *loadScript* amb l’argument “Error en el proceso”.
3. Promisifica la funció anterior: és a dir, fes que retorni una promesa. Recorda que hauràs de renombrar el callback com *resolved* i el errorcallback com *rejected*.
4. Crida la funció promisificada i consumeix la promesa que retorna amb els consumidors *then* i *catch* perquè tingui el mateix comportament que en la versió amb callbacks.

**Tasca 2**. En altres llenguatges de programació (C, per exemple), existeixen funcions de *sleep(ms)*, que interrompen l’execució del codi durant tants mil·lisegons. A els llenguatges d’alt nivell com JS no existeixen aquest tipus de funcions, perquè no hi està permés que cap funció sigui bloquejant.

Pero podem simular una funció com aquesta fent ús de promeses. Programa una funció promisificada (és a dir, que retorni una promesa), que només ha de fer el següent: resoldre la promesa al cap de x ms. Se la consumirà de la següent manera:

function sleep(miliseconds){

    (…)

}

sleep(3000)

.then(()=>{

    console.log('Sleep finalizado. Continua ejecución del programa...');

})