TSR: Pràctica 1. Sessió 1.
Execució de programes JavaScript

Treball a realitzar

Durant aquesta primera sessió has d'executar, entendre i ser capaç de raonar sobre diferents característiques del llenguatge JavaScript, sobretot si comparem aquest llenguatge amb algun llenguatge que coneguem més, com pot ser Java.

Treballarem sobre els següents aspectes:

1. Hola món en JavaScript
2. Àmbit de les variables
3. Tipus de les variables
4. Arguments de funcions
5. Classes en JavaScript
6. Clausures
7. Programació asíncrona mitjançant *callbacks*

Nota: Disposes de material de suport amb diversos documents i exemples que et poden servir per a familiaritzar-te amb el llenguatge JavaScript. Tot això està contingut en la **Pràctica 0: pràctica no guiada, no presencial**, que estimem que tot alumne ha de realitzar com a pas previ al desenvolupament de les pràctiques. La Pràctica 0, t'ha de permetre el treball amb els pròxims exemples de forma molt més senzilla. Allí es presenten múltiples exemples i explicacions de diversos conceptes que continuem treballant i ampliant en aquesta sessió de pràctiques. **Si no has fet la pràctica 0,** hauries de repassar els documents d'aqueixa pràctica ja que contenen material que et facilitarà el treball en aquesta pràctica i següents.

1.- Hola món en JavaScript

Amb aquest exemple repassem els elements mínims d'un programa en JavaScript.

1.1 Llig i executa el programa "ej1_HolaMundo.js"

- Observa l'ús de la directiva "use strict" al començament del programa.
- Observa que no hi ha funció "main". S'executa directament línia a línia tot el codi que proporcionem.
- Observa la diferència entre declarar una funció i executar-la.
- Raona tots els passos que s'executen i el que observes que s'imprimeix per la pantalla.

2.- Àmbit de les variables

Hem de conèixer les diferències entre utilitzar "let" i utilitzar "var" per a declarar variables. Hem de saber quan podem usar cadascuna d'elles i quines implicacions té. També hem de remarcar el que ocorre si no emprem "let" ni "var" sense emprar el mode estricte de JavaScript.

2.1 Llig i Executa el programa "ej2_AmbitoVariables.js" i **contesta** a les següents qüestions:

Qüestió 1. Observa que el programa modifica el valor d'una variable global dins de la funció “f1”. Podem modificar el valor de les altres variables globals? Quines diferències hi ha entre declarar variables globals amb “let” i amb “var”?

Qüestió 2. Observa que en la línia 24 s'imprimeix el valor de la variable “var_i”. Podem imprimir el valor de la variable “let_i”? Prova a fer-ho i raona el que observes. Raona quines diferències hi ha entre usar “let” i usar “var”.

Qüestió 3. Aquest programa no utilitza la directiva “use strict” al començament del programa. Afig-la i prova d'executar novament el programa. Què observes? Explica per a què serveix la directiva “use strict”.

Qüestió 4. Modifica el programa perquè funcione correctament usant la directiva “use strict”. Corregeix l'error que ha comès el programador per referenciar equivocadament la variable “local1_let”. Raona quins avantatges té per al programador l'ús de la directiva “use strict”.

3. Tipus de les variables

JavaScript és un llenguatge sense tipus. Aquest fet pot ser bastant confús en moltes situacions i és necessari programar sent conscients dels tipus de les variables i de les seues conversions.

3.1 Llig i executa el programa “ej3_TiposDeVariables.js”

3.2 Raona per què la resta funciona de manera diferent a la suma.

3.3 Consulta en Internet informació sobre el llenguatge “TypeScript”. Raona els motius que estan portant a la seua creixent implantació avui dia.

4. Arguments de funcions

JavaScript és un llenguatge prou flexible quant al nombre d'arguments de les funcions i els seus tipus. Aquesta característica és potent i al mateix temps pot portar a errors de programació.

4.1 Llig i executa el programa “ej4_Argumentos.js” i **contesta** a les següents qüestions:

Qüestió 1. Explica què fa el pseudo-vector “arguments”. Es pot accedir als diferents arguments mitjançant aquest pseudo-vector?

Qüestió 2. En imprimir result3, veiem “NaN”. Què significa NaN i per què veiem aquest resultat?

Qüestió 3. Què signifiquen els 3 punts suspensius en la crida a imprimir “resultv1” ?

Qüestió 4. Observa que s'imprimeix com a resultat “resultv2”. Per què obtenim aquest resultat?

resultv2: 1,2,3,4undefinedundefined

5. Classes en JavaScript

El suport a programació orientada a objectes de JavaScript és una miqueta primitiu, encara que ha millorat molt des de les versions de 2015. Avui dia es pot usar la sintaxi més recent i còmoda de classes o utilitzar la forma "antiga" de crear classes i instàncies. Totes dues formes coexisteixen.

5.1. Llig i executa el programa ej5-1_Clases.js

Observa com s'empra "this", de manera similar a com s'utilitza en altres llenguatges.
Observa com es declaren els atributs de les classes. Esborra l'atribut "nombre" i torna a executar el programa. Observa que la declaració d'atributs és completament opcional.
Observa com es declaren els constructors i com s'empra "super".

Aquest programa fa ús de la sintaxi més moderna de JavaScript. Encara és molt possible que el suport a objectes de JavaScript siga millorat en futures versions.

Encara que l'ús de classes siga recomanable en JavaScript, no s'aplicarà en aquesta assignatura. Per això, no es presenten altres exemples relacionats amb classes en aquesta pràctica.

6.- Clausures

Segurament un dels conceptes més importants de JavaScript (i d'altres llenguatges) són les clausures. Aquesta secció de pràctiques no pretén donar un curs complet de clausures, però sí donar unes pinzellades bàsiques del seu ús.

Per a simplificar l'explicació podem resumir dient que tota funció que s'executa en JavaScript té associada una clausura. La clausura d'una funció són les variables (i funcions) que es referencien des d'aquesta funció. Aquests símbols que es referencien des de la funció, romanen en memòria, mentre la funció estiga sent referenciada. El sentit pràctic el trobem quan la clausura conté **símbols que no siguen globals.**

El cas més senzill el tenim en una funció que no referencia a res. En aquest cas podem dir que la seua clausura és buida. Més habitualment es diu simplement que aquesta funció no té clausura o que no fa ús de clausures. El mateix ocorre si la clausura únicament referencia símbols globals, perquè aquests símbols sempre són accessibles i no és necessari cap "esforç per part del llenguatge" per a permetre l'accés a aquests símbols.

6.1.- Llig i executa el programa "ej6-1_Clausuras.js"

Observa que la funció f1 no té clausura.
Observa que la clausura de f2 és la variable x. Com es tracta d'un símbol global, realment no és una clausura en sentit estricte, perquè la funció pot accedir a aquesta variable global sense cap problema.
Observa la funció f3. En aquest cas tenim el **patró habitual** de les clausures. Una funció que retorna una altra funció. La funció f3 es diu **funció generadora** i la funció que retornem sol dir-se **funció clausura**, o simplement clausura, per a abreujar. La funció clausura retornada per f3 té com a clausura l'argument "arg" i la variable "i". Tots dos símbols són locals a la funció "f3" i per tant són visibles en la funció clausura, perquè estan en el seu àmbit. El suport a clausures de JavaScript mantindrà aquestes

variables en memòria mentre la funció clausura estiga referenciada. Observa com la funció clausura està referenciada en la variable “f”. Observa com perdura el valor de la variable “i” entre invocacions successives.

6.2.- Llig i executa el programa “ej6-2_Clausuras.js”

Este exemple il·lustra un cas d'ús habitual, sobretot en programari JavaScript per a navegadors. Observa com hi ha 3 segments de codi que fan pràcticament el mateix. El primer segment usa una variable global. El segon segment de codi usa una funció generadora i una clausura i d'aquesta manera evitem l'ús de variables globals. No obstant això creem un símbol global per a mantenir la funció. El tercer segment de codi és més complex, però més potent, perquè mitjançant clausures i funcions anònimes podem evitar l'ús de símbols globals.

Respon a la següent qüestió.
Qüestió 1. Per què pot resultar interessant emprar un patró com el descrit en aquest exemple quan fem programari que serà emprat com a biblioteca, des d'un navegador o des de nodejs?

6.3.- Llig i executa el programa “ej6-3_Clausuras.js”

Aquest exemple il·lustra una clausura que conté variables, arguments i funcions. Després d'executar i estudiar el codi, contesta la següent qüestió:

Qüestió 1. Quina és la clausura que retorna la funció generadora? Detalla les variables, arguments i funcions que formen part de la clausura.

6.4.- Llig i executa el programa “ej6-4_Clausuras.js”

Aquest exemple mostra una funció generadora que retorna una funció entre 2 possibles. Després d'executar i estudiar el codi, contesta les següents qüestions:

Qüestió 1. Quina és la clausura de la funció g0?

Qüestió 2. Quina és la clausura de la funció g1?

Qüestió 3. Quantes còpies en memòria hi ha de la variable “traza” ? Recorda que les clausures mantenen en memòria les variables a les quals referencien.

7.- Programació asincrònica mitjançant *callbacks*

La programació asincrònica proporciona una aproximació a la concurrència diferent a la que proporciona la programació clàssica multi-fil. És una aproximació que pot veure's com a complementària, més que com a alternativa.

En les classes de teoria s'aprofundirà en el **bucle d'esdeveniments**, com a mecanisme central a la programació asincrònica que trobem en JavaScript. Per tant, aquesta secció de pràctiques no substitueix al contingut de teoria, sinó més aviat, ens pot servir com una primera aproximació a uns certs aspectes pràctics.

Per a simplificar, podem dir que un programa en “nodejs”, pot **registrar manejadors d'esdeveniments**. Quan aquests esdeveniments succeïsquen, s'executaran els manejadors. És una cosa similar als manejadors d'interrupció dels sistemes operatius, o els manejadors d'esdeveniments que trobem en certes biblioteques gràfiques. (De fet, diverses biblioteques gràfiques que s'estudien en aquesta Universitat es basen en un bucle d'esdeveniments: AWT, Swing, JavaFX, etc).

L'esdeveniment més senzill de generar i de manejar segurament és el temps.

Amb la crida “setTimeout”, fem simultàniament 2 coses: instal·lem un manejador per a l'esdeveniment “timeout” i demanem que es produïsca l'esdeveniment “timeout” dins d'un determinat nombre de mil·lisegons. Més endavant, quan el temps que hàgem indicat vença, s'executarà aquest manejador de manera asincrònica.

7.1.- Llig i executa el programa ej7-1_Timeouts.js

Observa el que fa el programa i contesta les següents qüestions:

Qüestió 1. Per què veiem el missatge “cinco” abans del “cuatro”

Qüestió 2. Per què veiem “dos” abans que “tres” o “tres” abans que “dos”? Per a raonar aquesta qüestió executa diverses vegades el programa modificant el nombre d'iteracions del bucle perquè faça 100, 1000, 10000, 100000 iteracions.

7.2.- Llig i executa el programa ej7-2_Timeouts.js

El programa registra 10 manejadors d'esdeveniment i aconsegueix el final del programa.

Observa que el programa no acaba en imprimir el missatge final per la consola.

Observa el valor de variable que s'imprimeix..

Contesta les següents qüestions:

Qüestió 1. Per què sempre imprimeix 10?

Qüestió 2. Per què imprimeix un missatge cada segon?

Qüestió 3. Per què acaba el programa en imprimir 10 missatges per la consola? Per què no va acabar en executar l'última línia de codi del programa?

7.3.- Llig i executa el programa ej7-3_Timeouts.js

Aquest programa combina la gestió d'esdeveniments amb clausures. Amb aquest xicotet exemple podem apreciar que millora l'exemple anterior gràcies a les clausures.

El codi és breu, però és necessari estudiar-lo amb cura. Dedica-li diversos minuts. Si comprens aquest programa hauràs fet un pas important per a comprendre la manera de programar mitjançant JavaScript i nodejs.

Contesta a les següents qüestions:

Qüestió 1. Identifica la funció generadora i la funció de clausura. Quan i quantes vegades es crida la funció generadora? Quantes funcions de clausura es creen i qui les invoca?

Qüestió 2. Quin és el contingut de les clausures?

Qüestió 3. Quin avantatge ofereix en aquest exemple l'ús de les clausures?

7.4.- Llig i executa el programa ej7-4_Timeouts.js

Aquest exemple crea clausures sense utilitzar funcions generadores. Podem més aviat parlar d'un bloc generador de les clausures. Servisca aquest exemple per a recordar que les funcions generadores constitueixen un patró habitual per a la creació de clausures, però com aquest exemple il·lustra, no són estrictament necessàries.

Observa que el bloc generador de clausures declara una variable mitjançant “let”.

Contesta a la següent qüestió.

Qüestió 1. Modifica el programa perquè declare la variable dins del bloc “do” mitjançant “var”. Raona per què ara el funcionament és diferent.

7.5.- Llig i executa el programa ej7-5_Timeouts.js

Aquest exemple conté un programa una mica més complet que la resta. Podem observar clausures, un bucle de generació de clausures i proporciona una funció que serà notificada quan totes les clausures acaben. La manera de notificar aquesta fi, és mitjançant un *callback*.

Estudia i executa el codi per a entendre què fa i contesta a la següent qüestió.

Qüestió 1. Quan s'executarà la funció de *callback* proporcionada a la funció forkJoinAsync i des d’on s'executarà?