XDNATIS

ORSYS formation

Mécanismes avancés

Michael

XDNATIS

XDNATIS

ORSYS formation

Compétence visée :
Comprendre les mécanismes avancés de JS

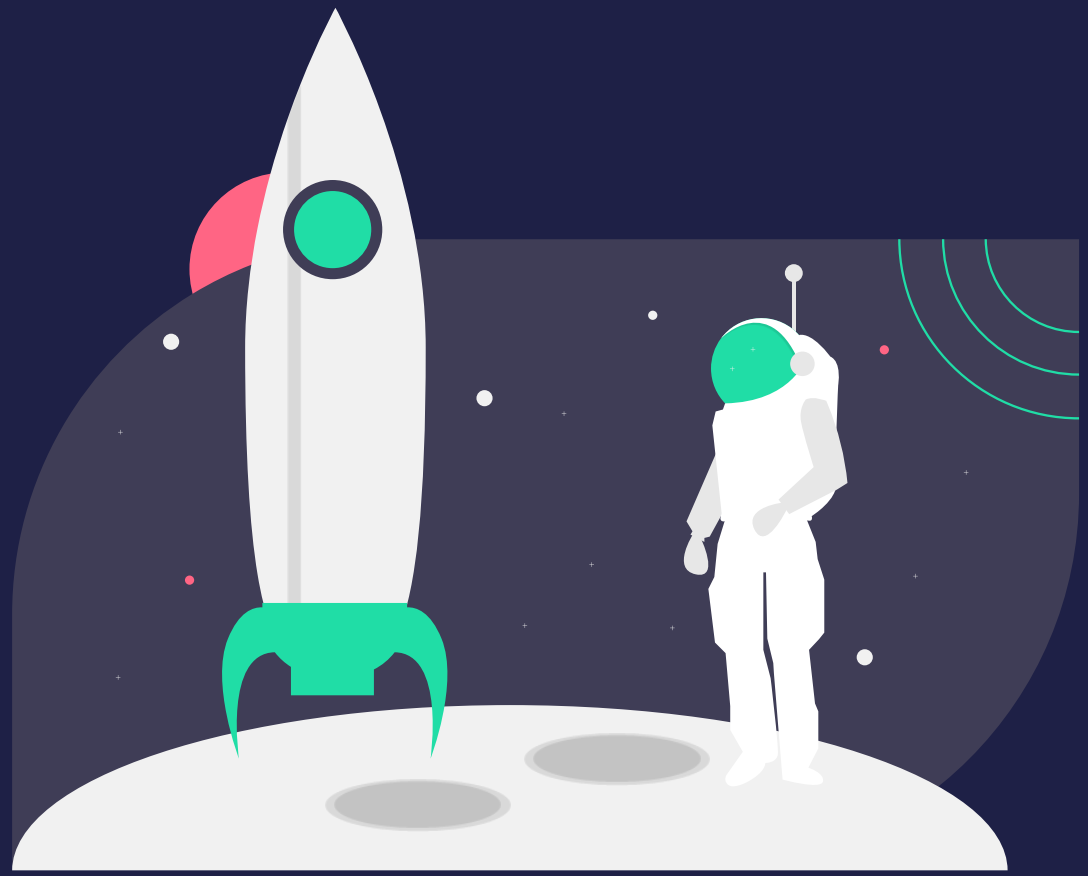
XWIKI SAS

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)



Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Le hoisting (en français, "hissage") est un terme que vous ne trouverez dans aucune prose de spécification normative avant l'ECMAScript® 2015.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Les déclarations de variables et de fonctions sont mises en mémoire pendant la phase de compilation, avant la phase d'exécution.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Le hissage a été conçu comme une façon générale de penser à la manière dont les contextes d'exécution (précisément, les phases de création et d'exécution) fonctionnent en JavaScript.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Le hissage fonctionne tout aussi bien avec d'autres types de données et d'autres variables.

Les variables peuvent être initialisées et utilisées avant d'être déclarées.

Mais elles ne peuvent pas être utilisées sans initialisation !

Avec ECMAScript 2015 (ES6), let et const remontera (hoisting) la déclaration variable au début de la portée (au début du bloc) mais pas l'initialisation.

Si on fait référence à une variable dans un bloc avant la déclaration de celle-ci avec let, cela entraînera une exception ReferenceError.

En effet, la variable est placée dans une « zone morte temporaire » entre le début du bloc et le moment où la déclaration est traitée. Autrement dit, la déclaration est bien remontée mais la variable ne peut pas être utilisée tant que l'affectation (qui n'est pas remontée) n'a pas été effectuée.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

La méthode JSON.stringify() convertit une valeur JavaScript en chaîne JSON. Optionnellement, elle peut remplacer des valeurs ou spécifier les propriétés à inclure si un tableau de propriétés a été fourni.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

La méthode JSON.parse() analyse une chaîne de caractères JSON et construit la valeur JavaScript ou l'objet décrit par cette chaîne. On peut éventuellement utiliser cette fonction avec un paramètre de modification permettant de traiter l'objet avant qu'il soit renvoyé.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Base64 est un schéma pour encoder des données binaires sous forme d'un texte au format ASCII grâce à la représentation de ces données en base 64. Le terme base64 vient à l'origine de l'encodage utilisé pour transférer certains contenus MIME.

La méthode btoa () crée une chaîne ASCII codée en Base64 à partir d'une chaîne binaire (c'est-à-dire un objet String dans lequel chaque caractère de la chaîne est traité comme un octet de données binaires).

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

La fonction atob () décode une chaîne de données qui a été codée en utilisant le codage Base64.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

PRATIQUE

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Normalement, le code d'un programme donné se déroule sans interruption, une seule chose se produisant à la fois. Si une fonction dépend du résultat d'une autre fonction, elle doit attendre que l'autre fonction se termine et retourne sa réponse, et jusqu'à ce que cela se produise, le programme entier est essentiellement bloqué

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Il est inutile de rester assis à attendre quelque chose alors que vous pouvez laisser une tâche se dérouler sur un autre cœur de processeur et être averti quand elle a terminé. Cela vous permet d'effectuer d'autres travaux en même temps, ce qui est la base de la programmation asynchrone.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Dans sa forme la plus élémentaire, JavaScript est un langage synchrone, bloquant et à un seul processus, dans lequel une seule opération peut être en cours à la fois.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Mais les navigateurs web définissent des fonctions et des API qui nous permettent d'enregistrer des fonctions qui ne doivent pas être exécutées de manière synchrone, mais qui peuvent être invoquées de manière asynchrone lorsqu'un événement quelconque se produit

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Attention, le asynchrone veut dire que l'on a pas besoin d'attendre la réponse d'un traitement avant d'en démarrer un autre, mais cela veut pas dire concurrent (faire les choses en parallèle) !

Le JavaScript reste mono-threadé (sauf exception avec les workers)

Cela signifie que vous pouvez laisser votre code faire plusieurs choses en même temps sans arrêter ou bloquer votre processus principal.

Il existe deux principaux types de code asynchrone que vous rencontrerez dans le code JavaScript : les anciens rappels et le code plus récent de type promesse

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Les callbacks asynchrones ou fonctions de rappels asynchrones sont des fonctions qui sont passées comme arguments lors de l'appel d'une fonction qui commencera à exécuter du code en arrière-plan.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Lorsque le code d'arrière-plan a fini de s'exécuter, il appelle la fonction de rappel pour vous faire savoir que le travail est terminé, ou pour vous faire savoir que quelque chose d'intéressant s'est produit.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

L'utilisation des callbacks est un peu démodée aujourd'hui, mais vous les verrez encore dans un certain nombre d'API plus anciennes encore couramment utilisées.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Les promesses sont le nouveau style de code asynchrone que vous verrez utilisé dans les API Web modernes.
Les promesses présentent certaines similitudes avec les anciennes fonctions de rappel.

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Il s'agit essentiellement d'un objet retourné auquel vous attachez des fonctions de rappel, plutôt que de devoir passer des callbacks dans une fonction.

Vous pouvez enchaîner plusieurs opérations asynchrones en utilisant plusieurs opérations .then(), en passant le résultat de l'une dans la suivante comme entrée ! ☺

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Les fonctions de rappel passées aux promesses sont toujours appelées dans l'ordre strict où ils sont placés dans la file d'attente des événements.

La gestion des erreurs est bien meilleure - toutes les erreurs sont traitées par un seul bloc .catch() à la fin du bloc, plutôt que d'être traitées individuellement à chaque niveau de la « pyramide ».

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

Les promises introduisent un paradigme de développement différent : event-driven development

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

La programmation événementielle est un paradigme de programmation dans lequel le flux du programme est déterminé par des événements tels que les actions de l'utilisateur (clics de souris, pressions sur les touches), les sorties de capteurs ou les messages transmis par d'autres programmes

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

CommonJS est un système de formatage de module. C'est un standard pour structurer et organiser le code JavaScript. CJS aide au développement d'applications côté serveur et son format a fortement influencé la gestion des modules de NodeJS.

Asynchronous Module Definition (AMD loader) est une spécification qui définit des modules JavaScript (techniquement chaque module est un fichier .js) qui pourront être ensuite chargés en parallèle en mode asynchrone, dans leur propre scope - donc sans conflit, en même temps que leurs dépendances, offrant ainsi une scalabilité accrue

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

La spécification ESM (EcmaScript Modules) introduite avec ES6 ou ES2015 décrit comment importer et exporter des modules en JavaScript. À présent il est possible d'utiliser ESM de façon native

Le Hoisting et la TDZ
1. hoisting
2. let et const vs var

La sérialisation
1. JSON
2. Base64
3. Pratique (stocker des données en localstorage)

La programmation asynchrone
1. Asynchrone ou concurrent ?
2. Callbacks
3. Promise
4. Event-driven development

Les modules
1. CommonJS
2. AMD avec requirejs
3. ESM
4. Pratique (client d'une API)

PRATIQUE

XNATIS

ORSYS formation

Pour aller plus loin

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using_promises