



JAVASCRIPT

Quizz d'entrainement #2

CDA 23016

[Notes]

liste des thèmes du questionnaire :

- Syntaxe et Types de Données
- Fonctions
- Objets et Prototypes
- Manipulation du DOM
- Événements
- Asynchrone (Promesses, Async/Await)
- Modules et Import/Export
- Gestion des Erreurs

[IMPORTANT]

plusieurs réponses sont possible sur certaines questions

Syntaxe et Types de Données

Question 1 : Comment déclarer et utiliser une variable de type string en JavaScript ?

- [] `let str = "Hello";`
- [] `const str = 'Hello';`
- [] `var str = "Hello";`
- [] `string str = "Hello";`

Question 2 : Quelle est la différence entre == et === en JavaScript ?

- [] == compare la valeur, === compare la valeur et le type.
- [] == compare le type, === compare la valeur.
- [] == et === sont identiques.
- [] == compare la valeur et le type, === compare seulement la valeur.

Question 3 : Comment vérifier si une variable est de type number en JavaScript ?

- [] `typeof variable === 'number'`
- [] `variable instanceof Number`
- [] `variable.isNumber()`
- [] `Number.isNumber(variable)`

Question 4 : Quelle est la différence entre un array et un object en JavaScript ?

- [] Un array est une liste ordonnée, un object est une collection de paires clé-valeur.
- [] Un array est une collection de paires clé-valeur, un object est une liste ordonnée.
- [] Un array et un object sont identiques.
- [] Un array est une fonction, un object est une liste.

Fonctions

Question 5 : Comment déclarer une fonction en JavaScript ?

- ☐ `function myFunction() {}`
- ☐ `let myFunction = function() {}`
- ☐ `const myFunction = () => {}`
- ☐ Toutes les réponses ci-dessus sont correctes.

Question 6 : Qu'est-ce qu'une fonction fléchée (arrow function) et comment l'utiliser ?

- [] `function arrowFunction() {}`
- [] `let arrowFunction = () => {}`
- [] `const arrowFunction = function() {}`
- [] `arrowFunction = () => {}`

Question 7 : Comment passer des paramètres par défaut à une fonction en JavaScript ?

- [] `function myFunction(param = 'default') {}`
- [] `function myFunction(param = default) {}`
- [] `function myFunction(param = defaultValue) {}`
- [] `function myFunction(param = 'defaultValue') {}`

Question 8 : Qu'est-ce qu'une fonction anonyme et comment l'utiliser ?

- [] `function() {}`

- [] `let anonymousFunction = function() {}`

- [] `const anonymousFunction = () => {}`

- [] `anonymousFunction = function() {}`

Question 9 : Comment utiliser la méthode apply pour appeler une fonction avec un contexte spécifique ?

- [] myFunction.apply(context, [args])

- [] myFunction.call(context, args)

- [] myFunction.bind(context, args)

- [] myFunction.context(args)

Objets et Prototypes

Question 10 : Comment créer un objet en JavaScript ?

- [] `let obj = {}`

- [] `const obj = new Object()`

- [] `var obj = Object.create()`

- [] Toutes les réponses ci-dessus sont correctes.

Question 11 : Qu'est-ce qu'un prototype en JavaScript et comment l'utiliser ?

```
- [ ] function MyObject() {} MyObject.prototype.method = function() {}
```

```
- [ ] let MyObject = { prototype: { method: function() {} } }
```

```
- [ ] const MyObject = function() { this.prototype = { method: function() {} } }
```

```
- [ ] var MyObject = { method: function() {} }
```

Question 12 : Comment ajouter une méthode à un prototype d'objet ?

- [] `MyObject.prototype.newMethod = function() {}`

- [] `MyObject.newMethod = function() {}`

- [] `MyObject.prototype = { newMethod: function() {} }`

- [] `MyObject.newMethod = new Function()`

Question 13 : Quelle est la différence entre Object.create et l'opérateur new ?

- ☐ Object.create crée un nouvel objet avec un prototype spécifié, new crée une instance d'une fonction constructeur.
- ☐ Object.create crée une instance d'une fonction constructeur, new crée un nouvel objet avec un prototype spécifié.
- ☐ Object.create et new sont identiques.
- ☐ Object.create est utilisé pour les objets, new est utilisé pour les fonctions.

Question 14 : Comment utiliser Object.assign pour fusionner des objets ?

- [] `Object.assign(target, source1, source2)`

- [] `Object.merge(target, source1, source2)`

- [] `Object.combine(target, source1, source2)`

- [] `Object.join(target, source1, source2)`

Manipulation du DOM

Question 15 : Comment sélectionner un élément par son ID en JavaScript ?

- [] `document.getElementById('id')`

- [] `document.querySelector('#id')`

- [] `document.getElementById('id')`

- [] `document.select('#id')`

Question 16 : Comment ajouter un nouvel élément au DOM ?

- [] `document.createElement('element')`

- [] `document.append('element')`

- [] `document.addElement('element')`

- [] `document.insert('element')`

Question 17 : Comment modifier le contenu textuel d'un élément ?

```
- [ ] element.textContent = 'new text'
```

```
- [ ] element.innerText = 'new text'
```

```
- [ ] element.setText('new text')
```

```
- [ ] element.content = 'new text'
```

Question 18 : Comment ajouter un écouteur d'événement à un élément ?

- [] `element.addEventListener('event', callback)`

- [] `element.addListener('event', callback)`

- [] `element.on('event', callback)`

- [] `element.listen('event', callback)`

Question 19 : Comment supprimer un élément du DOM ?

- [] `element.remove()`

- [] `element.delete()`

- [] `element.parentNode.removeChild(element)`

- [] `element.destroy()`

Événements

Question 20 : Comment ajouter un écouteur d'événement à un bouton en JavaScript ?

- [] `button.addEventListener('click', callback)`

- [] `button.onclick = callback`

- [] `button.click(callback)`

- [] `Toutes` les réponses ci-dessus sont correctes.

Question 21 : Quelle est la différence entre `addEventListener` et `onclick` ?

- ☐ `addEventListener` permet d'ajouter plusieurs écouteurs, `onclick` remplace l'écouteur existant.
- ☐ `addEventListener` remplace l'écouteur existant, `onclick` permet d'ajouter plusieurs écouteurs.
- ☐ `addEventListener` et `onclick` sont identiques.
- ☐ `addEventListener` est utilisé pour les événements, `onclick` est utilisé pour les fonctions.

Question 22 : Comment empêcher le comportement par défaut d'un événement ?

- [] `event.preventDefault()`

- [] `event.stopDefault()`

- [] `event.prevent()`

- [] `event.defaultPrevent()`

Question 23 : Comment arrêter la propagation d'un événement ?

- [] event.`stopPropagation()`

- [] event.`stopBubbling()`

- [] event.`halt()`

- [] event.`propagationStop()`

Question 24 : Comment déléguer des événements en JavaScript ?

- [] parent.**addEventListener**('event', callback)

- [] parent.**delegate**('event', callback)

- [] parent.**on**('event', callback)

- [] parent.**listen**('event', callback)

Asynchronisme (Promesses, Async/Await)

Question 25 : Qu'est-ce qu'une promesse en JavaScript et comment l'utiliser ?

```
- [ ] let promise = new Promise((resolve, reject) => {})
```

```
- [ ] let promise = Promise.create((resolve, reject) => {})
```

```
- [ ] let promise = new Promise((resolve, reject))
```

```
- [ ] let promise = Promise.new((resolve, reject) => {})
```

Question 26 : Comment utiliser async et await pour gérer des opérations asynchrones ?

- [] `async function myFunction() { await promise; }`

- [] `function myFunction() { async await promise; }`

- [] `async function myFunction() { return await promise; }`

- [] `function myFunction() { return async await promise; }`

Question 27 : Quelle est la différence entre Promise.all et Promise.race ?

- [] Promise.all attend que toutes les promesses soient résolues, Promise.race attend que la première promesse soit résolue.
- [] Promise.all attend que la première promesse soit résolue, Promise.race attend que toutes les promesses soient résolues.
- [] Promise.all et Promise.race sont identiques.
- [] Promise.all est utilisé pour les promesses, Promise.race est utilisé pour les fonctions.

Question 28 : Comment gérer les erreurs avec async et await ?

```
- [ ] try { await promise; } catch (error) {}
```

```
- [ ] async { await promise; } catch (error) {}
```

```
- [ ] try { return await promise; } catch (error) {}
```

```
- [ ] async { return await promise; } catch (error) {}
```

Question 29 : Comment créer une promesse qui se résout après un certain délai ?

- [] `new Promise(resolve => setTimeout(resolve, 1000))`

- [] `Promise.delay(1000)`

- [] `new Promise(resolve => setTimeout(() => resolve(), 1000))`

- [] `Promise.timeout(1000)`

Modules et Import/Export

Question 30 : Comment exporter une fonction depuis un module en JavaScript ?

- [] `export function myFunction() {}`

- [] `module.exports = myFunction`

- [] `export myFunction = function() {}`

- [] `exports.myFunction = function() {}`

Question 31 : Comment importer une fonction depuis un module en JavaScript ?

- [] `import { myFunction } from 'module'`

- [] `const myFunction = require('module')`

- [] `import myFunction from 'module'`

- [] `let myFunction = import('module')`

Question 32 : Quelle est la différence entre export default et export nommé ?

- [] export default exporte une seule valeur par défaut, export nommé exporte plusieurs valeurs.
- [] export default exporte plusieurs valeurs, export nommé exporte une seule valeur par défaut.
- [] export default et export nommé sont identiques.
- [] export default est utilisé pour les fonctions, export nommé est utilisé pour les variables.

Question 33 : Comment utiliser des modules ES6 dans un projet JavaScript ?

- [] `import 'module'`

- [] `require('module')`

- [] `import * as module from 'module'`

- [] `const module = import('module')`

Question 34 : Comment gérer les dépendances avec des modules en JavaScript ?

- [] `import { myFunction } from 'module'`

- [] `const dependency = require('module')`

- [] `import dependency from 'module'`

- [] `let dependency = import('module')`

Gestion des Erreurs

Question 35 : Comment utiliser try, catch et finally pour gérer les erreurs en JavaScript ?

- [] `try { code } catch (error) { handleError } finally { cleanup }`

- [] `try { code } catch (error) { handleError }`

- [] `try { code } finally { cleanup }`

- [] `catch (error) { handleError } finally { cleanup }`

Question 36 : Qu'est-ce qu'une erreur de référence (ReferenceError) et comment la gérer ?

- ☐ Une erreur qui se produit lorsqu'une variable n'est pas définie.
- ☐ Une erreur qui se produit lorsqu'une fonction n'est pas définie.
- ☐ Une erreur qui se produit lorsqu'un objet n'est pas défini.
- ☐ Une erreur qui se produit lorsqu'une méthode n'est pas définie.

Question 37 : Comment lancer une erreur personnalisée en JavaScript ?

- [] `throw new Error('message')`

- [] `throw 'message'`

- [] `throw new Exception('message')`

- [] `throw new CustomError('message')`

Question 38 : Comment gérer les erreurs asynchrones avec async et await ?

```
- [ ] try { await promise; } catch (error) {}
```

```
- [ ] async { await promise; } catch (error) {}
```

```
- [ ] try { return await promise; } catch (error) {}
```

```
- [ ] async { return await promise; } catch (error) {}
```


Question 39 : Comment utiliser `console.error` pour afficher des messages d'erreur ?

- [] `console.error('message')`

- [] `console.log('error: message')`

- [] `console.warn('message')`

- [] `console.debug('message')`

Question 40 : Comment utiliser les paramètres rest et spread en JavaScript ?

- [] `function myFunction(...args) {}`

- [] `function myFunction(args...) {}`

- [] `function myFunction(args) {}`

- [] `function myFunction(...rest) {}`

Question 41 : Qu'est-ce que le let et const et comment les utiliser ?

- [] let est utilisé pour les variables mutables, const pour les variables immuables.
- [] let est utilisé pour les variables immuables, const pour les variables mutables.
- [] let et const sont identiques.
- [] let est utilisé pour les fonctions, const pour les objets.

Questions de Code

Question 42 : Écrivez une fonction qui prend un tableau de nombres et retourne la somme de tous les éléments.

```
function sumArray(arr) {  
  // Votre code ici  
}  
  
console.log(sumArray([1, 2, 3, 4, 5])); // Doit afficher 15
```

Question 43 : Écrivez une fonction fléchée qui prend deux nombres et retourne leur produit.

```
const multiply = (a, b) => {  
  // Votre code ici  
};  
  
console.log(multiply(3, 4)); // Doit afficher 12
```

Question 44 : Écrivez une fonction qui utilise `Object.assign` pour fusionner deux objets.

```
const obj1 = { a: 1, b: 2 };
const obj2 = { b: 3, c: 4 };

function mergeObjects(obj1, obj2) {
  // Votre code ici
}

console.log(mergeObjects(obj1, obj2)); // Doit afficher { a: 1, b: 3, c: 4 }
```

Question 45 : Écrivez une fonction qui ajoute un écouteur d'événement à un bouton et affiche un message lorsque le bouton est cliqué.

```
function addClickListener(button) {  
    // Votre code ici  
}  
  
const button = document.createElement('button');  
button.textContent = 'Click me';  
document.body.appendChild(button);  
addClickListener(button);
```

Question 46 : Écrivez une fonction qui utilise `async` et `await` pour attendre une seconde avant de retourner un message.

```
async function delayMessage() {  
  // Votre code ici  
}  
  
delayMessage().then(message => console.log(message)); // Doit afficher "Message after 1 second" après 1 seconde
```


bref. c'est fini

Created with markdown & Marp by Jérôme BOEBLON