Bing

D'accord, je vais essayer de générer un QCM de 8 questions sur les variables en JavaScript, avec un niveau difficile. Voici le QCM:

1. Quelle est la différence entre `var` et `let` pour déclarer une variable?
   * A) `var` crée une variable globale ou locale à la fonction, tandis que `let` crée une variable locale au bloc.
   * B) `var` crée une variable locale au bloc, tandis que `let` crée une variable globale ou locale à la fonction.
   * C) `var` et `let` sont équivalents, sauf que `let` est plus récent et plus recommandé.
   * D) `var` et `let` sont équivalents, sauf que `var` est plus récent et plus recommandé.
   * \*\*Réponse: A\*\*
   * \*\*Feedback: B est faux, car `var` ne respecte pas la portée des blocs. C et D sont faux, car `var` et `let` ont des comportements différents en termes de hoisting, de redéclaration et de temporal dead zone.\*\*
2. A) `var` crée une variable globale ou locale à la fonction, tandis que `let` crée une variable locale au bloc.
3. B) `var` crée une variable locale au bloc, tandis que `let` crée une variable globale ou locale à la fonction.
4. C) `var` et `let` sont équivalents, sauf que `let` est plus récent et plus recommandé.
5. D) `var` et `let` sont équivalents, sauf que `var` est plus récent et plus recommandé.
6. \*\*Réponse: A\*\*
7. \*\*Feedback: B est faux, car `var` ne respecte pas la portée des blocs. C et D sont faux, car `var` et `let` ont des comportements différents en termes de hoisting, de redéclaration et de temporal dead zone.\*\*
8. Quelle est la valeur de `x` après l'exécution du code suivant?

var x = 1;

function foo() {

var x = 2;

console.log(x);

}

foo();

console.log(x);

- A) 1

- B) 2

- C) undefined

- D) ReferenceError

- \*\*Réponse: A\*\*

- \*\*Feedback: La variable `x` déclarée dans la fonction `foo` est locale à la fonction, et n'affecte pas la variable globale `x`. Donc, le code affiche 2 puis 1.\*\*

3. Quelle est la valeur de y après l'exécution du code suivant?

var x = 1;

var y = x;

x = 2;

console.log(y);

- A) 1

- B) 2

- C) undefined

- D) ReferenceError

- \*\*Réponse: A\*\*

- \*\*Feedback: Les variables primitives comme les nombres sont copiées par valeur en JavaScript. Donc, quand on affecte `x` à `y`, on crée une nouvelle variable qui contient la même valeur que `x`, mais qui n'est pas liée à `x`. Donc, quand on change la valeur de `x`, cela n'affecte pas la valeur de `y`.\*\*

4. Quelle est la valeur de z après l'exécution du code suivant?

var x = {name: "Alice"};

var y = x;

x.name = "Bob";

console.log(y.name);

- A) "Alice"

- B) "Bob"

- C) undefined

- D) ReferenceError

- \*\*Réponse: B\*\*

- \*\*Feedback: Les variables non primitives comme les objets sont copiées par référence en JavaScript. Donc, quand on affecte `x` à `y`, on crée une nouvelle variable qui pointe vers le même objet que `x`, et qui est donc liée à `x`. Donc, quand on change la propriété de l'objet référencé par `x`, cela affecte aussi l'objet référencé par `y`.\*\*

5. Quelle est la valeur de a après l'exécution du code suivant?

console.log(a);

var a = 1;

- A) 1

- B) undefined

- C) null

- D) ReferenceError

- \*\*Réponse: B\*\*

- \*\*Feedback: Ce code illustre le phénomène de hoisting, qui consiste à remonter les déclarations de variables au début du scope. Donc, le code est équivalent à:

var a;

console.log(a);

a = 1;

Donc, la variable a existe, mais n'a pas encore de valeur définie quand on l'affiche.\*\*

1. Quelle est la valeur de `b` après l'exécution du code suivant?

console.log(b);

let b = 1;

- A) 1

- B) undefined

- C) null

- D) ReferenceError

- \*\*Réponse: D\*\*

- \*\*Feedback: Ce code illustre la notion de temporal dead zone, qui concerne les variables déclarées avec `let` ou `const`. Ces variables ne sont pas hoistées, mais elles exist