JavaScript Révisions

Bonjour, Demo User #447

Votre copie a bien été rendue. Merci !

Note obtenue: 1 / 54

Sign in

Les bases - QCM

Question 1 (1 points)

Comment ouvrir la console JavaScript dans Google Chrome ?

- ○ En tapant "console"
- ○ En appelant le prof
- ○ En demandant gentiment à Siri
- ◉ En pressant Cmd-Alt-J ou Ctrl-Shift-J

Réponse: *en pressant Cmd-Alt-J ou Ctrl-Shift-J*

Question 2 (0 points)

Que retourne `typeof` quand il est appliqué sur `"bonjour"` ?

- ○ "string"
- ○ string
- ○ "object"
- ○ undefined

Réponse: `"string"`. Pour le vérifier, taper `typeof "bonjour";` dans la console JavaScript.

Question 3 (0 points)

Types de valeurs en JavaScript. Quel est l'intrus ?

- string
- boolean
- decimal
- number

Réponse: `decimal` était l'intrus. En JavaScript, les nombres décimaux sont compris dans le type `number`. Vérifier en tapant `typeof 4.5;` dans la console.

Question 4 (0 points)

Comment créer une variable en JavaScript ?

- maVariable;
- var maVariable;
- x = 0;
- maVariable = 'bonjour';

Réponse: `var maVariable;` Pour créer une variable, il faut utiliser le mot-clé `var`. Après, il est possible de changer la valeur de cette variable sans avoir à utiliser `var`. Il est aussi possible d'affecter une valeur à cette variable au moment de sa création: `var maVariable = 4;`

Question 5 (0 points)

Quel est le type de cette variable:

```
var maVariable = 3.2;
```

- ○ string
- ○ number
- ○ decimal
- ○ boolean

Réponse: en Javascript, les nombres décimaux sont aussi de type `number`.

Question 6 (0 points)

Quel est le type de cette variable:

```
var maVariable = '3.2';
```

- ○ string
- ○ number
- ○ decimal
- ○ boolean

Réponse: la valeur est entourée d'apostrophes => c'est une chaîne de caractères (`string`).

Question 7 (0 points)

Comment afficher la valeur d'une variable appelée `maVariable` depuis la console ?

- maVariable;
- var maVariable;
- maVariable?
- show maVariable

Réponse: `maVariable;` Dans la console JavaScript, il suffit de taper le nom du variable pour afficher sa valeur, de la même façon que taper `1+1` provoquera l'affichage de `2`. Le point virgule n'est pas imposé par la console JavaScript, mais c'est une convention à suivre.

Question 8 (0 points)

Comment changer la valeur d'une variable existante ? (déjà créée)

- var maVariable = 4;
- maVariable = 4;
- maVariable(4);
- 4 = maVariable;

Réponse: `maVariable = 4;` Le mot-clé `var` n'est à utiliser que lorsque la variable n'a pas encore été créée. L'usage des parenthèses dans `maVariable(4);` provoque l'appel d'une fonction appelée `maVariable` en passant la valeur `4` en paramètre. Enfin, l'opérateur d'affectation stocke la valeur à droite du `=` dans la variable à gauche du `=`, donc `4 = maVariable` n'a pas de sens car `4` n'est pas une variable.

Question 9 (0 points)

Si j'ai créé une variable dont la valeur est un nombre, que se passera-t-il si je lui affecte ensuite une chaine de caractères ?

- ○ erreur, car le type est différent.
- ○ erreur, car on ne peut pas changer la valeur d'une variable.
- ○ la valeur de la variable va être remplacée par la chaine de caractères.
- ○ les deux valeurs vont être concaténées.

Réponse: *la valeur de la variable va être remplacée par la chaine de caractères.* L'affectation consiste à utiliser l'opérateur `=` pour remplacer la valeur d'une variable par une autre valeur. Pour effectuer une concaténation, il faut utiliser l'opérateur `+`, et non l'opérateur d'affectation. JavaScript est un langage faiblement typé, il est donc possible d'affecter une valeur de n'importe quel type à n'importe quelle variable.

Question 10 (0 points)

En respectant les conventions indiquées en cours, quelle affectation faut-il exécuter pour que `J'ai tout compris !` s'affiche à l'écran ?

```
alert(message);
```

- ○ message = "J'ai tout compris !";
- ○ message = 'J"ai tout compris !';
- ○ message = 'J\'ai tout compris !';
- ○ message = J'ai tout compris !

Réponse: `'J\'ai tout compris !'` car nous utilisons des apostrophes autour des chaînes de caractères, et antislash pour afficher des apostrophes à l'intérieur de chaînes de caractères.

Conditions - QCM

Question 11 (0 points)

Que vaut cette expression de comparaison de valeur ?

```
3.2 === '3.2'
```

- ○ true
- ○ false
- ○ undefined
- ○ c'est une affectation

Réponse: le triple égal est une comparaison stricte: elle renvoie `false` ici car les deux valeurs ne sont pas du même type.

Question 12 (0 points)

Quelle section va être exécutée, si on exécute le code suivant ?

```
var nb = 2;
if (nb === 1) {
  // A
} else {
  // B
}
```

- ○ A
- ○ B
- ○ A et B
- ○ aucune

Réponse: B, car l'expression d'égalité `nb === 1` de la premiere condition est fausse, donc ce sont les intructions rattachées à l'alternative par défaut (`else`) qui sont exécutées.

Question 13 (0 points)

Quelle section de va être exécutée, si on exécute le code suivant ?

```
var nb = 2;
if (nb === 2) {
  // A
} else if (nb > 1) {
  // B
} else {
  // C
}
```

- ○ A
- ○ B
- ○ A et B
- ○ A, B et C

Réponse: A. Une seule des trois alternatives peut s'exécuter, car elles sont liées par des `else`. Sachant que les conditions sont évaluées de haut en bas, et que la première expression est vraie, c'est donc la section A qui va s'exécuter.

Question 14 (0 points)

À quoi ressemblerait l'arbre de décision correspondant à ce code:

```
var reponse = prompt('as-tu faim ?')
if (reponse === 'oui') {
  var reponse2 = prompt('aimes-tu les burgers ?');
  if (reponse2 === 'oui') {
    alert('alors je t\'en offre un !');
  } else {
    alert('dommage !');
  }
} else {
  alert('désolé');
}
```

- ○ une boîte et deux branches
- ○ deux boîtes de même niveau
- ○ une boîte de niveau 1, et une boîte de niveau 2
- ○ une boîte et trois branches

Réponse: Une première boîte (niveau 1) représente la question `as-tu faim ?`, et a deux branches: `oui` et *autre*. Liée à la première branche, une deuxième boîte (niveau 2) représente la question `aimes-tu les burgers ?`, et a elle-aussi deux branches: `oui` et *autre*.

Question 15 (0 points)

Pourquoi faut-il éviter d'utiliser les opérateurs `==` et `!=` ?

- ○ car il vaut mieux utiliser une affectation =
- ○ car ils sont trop stricts
- ○ car ils sont trop laxistes
- ○ var === et !== sont plus lisibles

Réponse: Ils sont trop laxistes, dans le sens où deux valeurs de types différents (ex: `1` et `'1'`) peuvent être vus comme égaux par l'opérateur `==`. Cet excès de tolérance peut occasionner des comportements imprévus qui font perdre beaucoup de temps à diagnostiquer et à corriger. Idem pour `!=`.

Conditions - Code

Question 16 (0 points)

Implémenter une condition qui affecte `'egal'` à une variable `resultat` seulement si une autre variable `nombre` vaut strictement `4`. Indenter correctement.

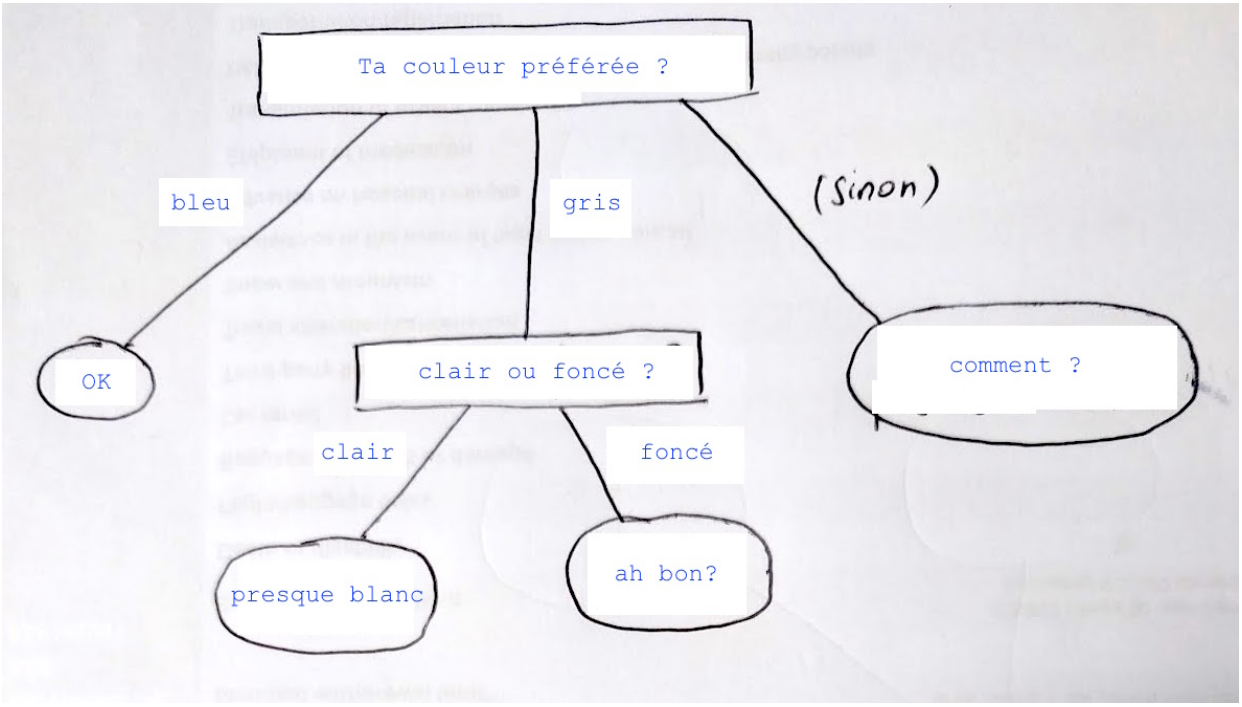
Saisissez votre code Javascript ici

Solution:

```
if (nombre === 4) {
  resultat = 'egal';
}
```

Question 17 (0 points)

Implémenter un chat-bot à partir de l'arbre de décision suivant:



Comme vu et pratiqué en cours:

- Les questions posées par l'ordinateur sont représentées par des rectangles, et sont à implémenter à l'aide de la fonction `prompt()`.
- Les réponses comprises par l'ordinateur sont écrites à côté de chaque branche, et sont à implémenter à l'aide de conditions.
- Les messages à afficher par l'ordinateur sont représentés par des cercles, et sont à implémenter à l'aide de la fonction `alert()`.

Vous serez noté(e) sur:

- le respect **à la lettre** du texte des questions et des réponses (espaces, accents, et majuscules/minuscules compris).
- le respect des règles d'indentation et autres conventions vues en cours. (ex: 2 espaces par niveau d'indentation)
- le bon fonctionnement de votre code, sans erreurs, depuis la console JavaScript de Google Chrome, pour chacun des cas illustrés dans l'arbre de décision.

Saisissez votre code Javascript ici

Solution:

```
var reponse = prompt('Ta couleur préférée ?');
if (reponse === 'bleu') {
  alert('OK');
} else if (reponse === 'gris') {
  var reponse2 = prompt('clair ou foncé ?');
  if (reponse2 === 'clair') {
    alert('comme le ciel');
  } else if (reponse2 === 'foncé') {
    alert('ah bon?');
  }
} else {
  alert('je connais pas');
}
```

Algo / Boucles - QCM

Question 18 (0 points)

Combien de fois les instructions vont-elles être exécutées ?

```
for ( var i = 0; i < 4; i++ ) {
  // instructions
}
```

- ○ 0 fois
- ○ 1 fois
- ○ 3 fois
- ○ 4 fois

Les instructions vont être exécutées pour chaque valeur de `i` entre `0` et `4` (non compris), soit 0, 1, 2, puis 3.

Ce qui fait 4 itérations.

Question 19 (0 points)

Combien de fois les instructions vont-elles être exécutées ?

```
for ( var i = 3; i >= 1; i-- ) {
  // instructions
}
```

- ○ 0 fois
- ○ 1 fois
- ○ 3 fois
- ○ 4 fois

Les instructions vont être exécutées pour chaque valeur de `i` entre `3` et `1` (compris), soit 3, 2, puis 1.

Ce qui fait 3 itérations.

Algo / Boucles - Code

Question 20 (0 points)

Implémenter un programme de moins de 4 lignes qui affiche 50 fois `'Bonjour!'` dans la console. Respecter les conventions et règles d'indentation vues en cours.

Saisissez votre code Javascript ici

Solution:

```
for ( var i = 0; i < 50; i++ ) {
  console.log('Bonjour!');
}
```

Fonctions - QCM

Question 21 (0 points)

```
function maFonction(param) {
  return param + 2;
}
```

Ceci est:

- ○ un appel de fonction
- ○ une définition de fonction
- ○ une affectation de fonction
- ○ une fonction qui ne fonctionne pas

C'est une définition de fonction.

On la reconnait à l'usage du mot clé `function` et des accolades entourant le code qui sera exécuté lorsque cette fonction sera appelée.

Question 22 (0 points)

```
maFonction(4);
```

Ceci est:

- ○ un appel de fonction
- ○ une définition de fonction
- ○ une affectation de fonction
- ○ une fonction qui ne fonctionne pas

C'est un appel de fonction.

Un appel de fonction = le nom de la fonction, suivi par les paramètres entre parenthèses. Sans le mot clé `function`.

Cette instruction va exécuter le code défini dans la fonction, et affecter les valeurs fournies à chaque paramètre.

Question 23 (0 points)

```
// cette fonction concatène un zéro à la fin de la valeur passée en paramè
function maFonction(param) {
  return param + '0';
}
```

Comment savoir si cette fonction fonctionne bien ? (c.a.d. sans bug)

- ○ il suffit de la copier-coller dans la console
- ○ il faut taper maFonction dans la console
- ○ vérifier que le test passe: maFonction(1) === '10';
- ○ vérifier que maFonction(1) renvoie bien true

Pour vérifier le bon fonctionnement il faut définir et exécuter des tests unitaires.

Ceux-ci permettent de comparer le résultat attendu d'une fonction, à celui effectivement retourné par l'implémentation actuelle de cette fonction.

`maFonction(1) === '10';` est un bon test unitaire car son exécution retourne `true` si la fonction retourne le résultat attendu (`10`) lorsqu'on lui passe `1` en paramètre.

Question 24 (0 points)

Supposons que nous avons défini une fonction `doubler()` qui retourne le double du nombre passé en paramètre, lors de son appel.

Que se passe-t-il si on exécute l'instruction suivante:

```
var maVariable = doubler(3);
```

- ○ le résultat va être affecté à maVariable
- ○ le résultat va s'afficher dans la console
- ○ maVariable contient la définition de la fonction
- ○ maVariable contient l'appel de la fonction

Il s'agit ici d'un appel de fonction. De la même façon que pour une opération élémentaire (ex: `2 + 2`), tout appel de fonction sera remplacé par la valeur retourné par l'exécution de cette fonction.

Ici, le résultat de l'exécution de la fonction `doubler` avec le paramètre `3` , soit la valeur `6` , va être affectée à `maVariable` .

Fonctions - Code

Question 25 (0 points)

Définir une fonction `soustraire` qui retourne le résultat de la soustraction `a - b`, `a` et `b` étant des paramètres de cette fonction.

Respecter les conventions et règles d'indentation vues en cours.

Saisissez votre code Javascript ici

Solution:

```
function soustraire(a, b) {
  return a - b;
}
```

Question 26 (0 points)

Définir une fonction `somme` qui retourne la somme des trois nombres passés en paramètres.

Exemple d'appel: `somme(1, 3, -2);` doit retourner `2`.

Saisissez votre code Javascript ici

Solution:

```
function somme(a, b, c) {
  return a + b + c;
}
```

Question 27 (0 points)

Définir une fonction `repeter` qui affiche `n` fois `'Bonjour!'` dans la console, puis qui retourne `n`, `n` étant un paramètre de cette fonction.

Respecter les conventions et règles d'indentation vues en cours.

Saisissez votre code Javascript ici

Solution:

```
function repeter(n) {
  for (var i = 0; i < n; i++) {
    console.log('Bonjour!');
  }
  return n;
}
```

Question 28 (0 points)

Définir une fonction `sommeIntervalle` qui retourne la somme de tous les nombres entiers compris entre les nombres `premier` et `dernier` (compris) passés en paramètres.

Exemple d'appel: `sommeIntervalle(2, 5);` doit retourner `14` (résultat de `2 + 3 + 4 + 5`).

Saisissez votre code Javascript ici

Solution:

```
function sommeIntervalle(premier, dernier) {
  var somme = 0;
  for (var i = premier; i <= dernier; i++) {
    somme = somme + i;
  }
  return somme;
}
```

Tableaux - Code

Question 29 (0 points)

Créez une variable `nombres` de type tableau et contenant les nombres `1`, `2` et `3`.

Saisissez votre code Javascript ici

Solution:

```
var nombres = [1, 2, 3];
```

Question 30 (0 points)

Vous disposez d'une variable `fruits` contenant un tableau de chaînes de caractères.

Saisissez le code JavaScript pour créer une variable `troisieme` et lui affecter la valeur du 3ème élément de ce tableau.

Saisissez votre code Javascript ici

Solution:

```
var troisieme = fruits[2];
```

Question 31 (0 points)

Vous disposez d'une variable `fruits` contenant un tableau de chaînes de caractères.

Saisissez le code JavaScript permettant de retirer le dernier élément de ce tableau, et d'afficher la valeur de cet élément dans la console.

Saisissez votre code Javascript ici

Solution:

```
console.log(fruits.pop());
```

Question 32 (0 points)

Définir une fonction `tableauContient` qui prend deux paramètres:

- `tableau` : un tableau de chaînes de caractères
- `chaine` : une chaîne de caractères

...et retourne:

- `false` si la valeur `chaine` n'a pas été trouvée dans le tableau `tableau` ,
- ou le premier indice (à partir de 0) auquel a été trouvé la valeur `chaine` dans le tableau `tableau` .

Exemples d'appels:

- `tableauContient(['a', 'b', 'c'], 'b');` doit retourner `1` .
- `tableauContient(['a', 'b', 'c'], 'd');` doit retourner `false` .

Saisissez votre code Javascript ici

Solution:

```
function tableauContient(tableau, chaine) {
  var indice = tableau.indexOf(chaine);
  if (indice === -1) {
    return false;
  } else {
    return indice;
  }
}
```